TANGENT-BALL TECHNIQUES FOR SHAPE PROCESSING

A Dissertation
Presented to
The Academic Faculty

by

Brian Scott Whited

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
December 2009

TANGENT-BALL TECHNIQUES FOR SHAPE PROCESSING

Approved by:

Professor Jarek Rossignac,
Committee Chair
College of Computing
Georgia Institute of Technology

Professor Jarek Rossignac, Advisor
College of Computing
Georgia Institute of Technology

Professor Karen Liu
College of Computing
Georgia Institute of Technology

Dr. Maryann Simmons
Technology and R&D
Walt Disney Animation Studios

Dr. Greg Slabaugh
Research and Development
Medicsight, PLC

Professor Greg Turk
College of Computing
Georgia Institute of Technology

Date Approved: 28 October 2009

ACKNOWLEDGEMENTS

I wish to thank, first and foremost, my advisor Professor Jarek Rossignac, whose guidance, patience and attention insured that I never stopped pushing forward. If it weren't for him, I would never have known that a Ph.D. was something I wanted.

I also express gratitude to Dr. Chazal and Dr. Lieutier for their contributions and suggestions pertaining to the *ball-map* [24], *b-morph* [126], and *relative blending* [127]. I thank Dr. Greg Slabaugh for his time and mentoring during my two internships at Siemens Corporate Research and his contributions to *pearling* [125][124]. From my time at SCR, I'd also like to acknowledge Dr. Tong Fang, Dr. Gozde Unal, and Dr. Sergei Azernikov for their suggestions and collaborations. I thank Dr. Maryann Simmons for her mentoring and guidance during my internship at Walt Disney Animation Studios and also her contributions to the ongoing research of animation inbetweening. Also for their collaborations on animation inbetweening, I thank lead clean-up artist Rachel Bibb from WDAS and Dr. Joe Marks, Chino Noris, Dr. Bob Sumner, and Dr. Markus Gross from Disney Research.

I would also like to thank committee members Dr. Greg Turk and Dr. Karen Liu for their suggestions and advice throughout my Ph.D. career and their organization of the weekly Geometry group meetings. Throughout my years in the MAGIC lab, I have enjoyed studying and collaborating with fellow lab-mates Jason, Lorenzo, Alex, Justin and Topraj as well as Chris, Yuting, and Sumit and others in the graphics group.

And finally, and most importantly, I thank my parents, grandparents and my sister Steph for always supporting me and believing in me in everything I do. I also thank my friends, especially Diane, for always listening to my complaints, gripes, and

less-than-exciting tales of research.

This work has been partially supported by NSF grant 0811485, research gifts from the Walt Disney Corporation and Siemens Corporate Research, and also a seed grant from the GVU at Georgia Tech.

NOTATION

S	Set or curve S
$!S$	Complement of set S
iS	Interior of set S (Open set bounded by a curve loop S)
eS	Exterior of set S (Open set not bounded by a curve loop S)
bS	Closed boundary (curve or surface) of set S
cS	Closure of set S
S^r	Dilation of S by radius r
S_r	Set S eroded by r
$A \cup B$	Boolean union of two sets A and B
$A \cap B$	Boolean intersection of two sets A and B
$A \Delta B$	Symmetric difference of sets A and B
moat $X(A, B)$	$(A \cup iA)\Delta(B \cup iB) \cup A \cup B$ for closed loop curves A and B
	$i(A \cup B) \cup A \cup B$ for open endpoint-aligned curves A and B
$R_B(A)$	Relative blending of A with respect to control shape B
$\mathbf{p}$	Cartesian point $\mathbf{p}$
$\overrightarrow{\mathbf{v}}$	Vector $\overrightarrow{\mathbf{v}}$
$R(\overrightarrow{\mathbf{v}})$	Vector $\overrightarrow{\mathbf{v}}$ rotated by π
$\lvert\overrightarrow{\mathbf{v}}\rvert$	Magnitude of vector $\mathbf{v}$
$\overrightarrow{\mathbf{ab}}$	Vector line segment $\mathbf{b} - \mathbf{a}$
$\widehat{\mathbf{v}}$	Unit vector $\widehat{\mathbf{v}}$
$\widehat{\mathbf{N}}_P(\mathbf{p})$	Unit normal to set P at point $\mathbf{p}$
$\triangle\mathbf{abc}$	Triangle with vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}$

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Shape processing comprises a set of theoretical and algorithmic tools for creating, measuring and modifying digital representations of shapes (curves in 2D or surfaces in 3D). Such tools are of paramount importance to many disciplines of computer graphics, including modeling, animation, visualization, and image processing. Many applications of shape processing can be found in the entertainment and medical industries. Examples of shape processing include:

1. Computing a point-to-point correspondence between different shapes
2. Identifying and filtering (smoothing) a shape's sharp features
3. Segmenting a shape into components relevant to a particular application domain
4. Measuring the local discrepancy (difference) between shapes
5. Computing an animation that morphs (smoothly interpolates) from one shape to another
6. Combining shapes through Boolean (union, intersection) or morphological (grow, shrink, Minkowski sum) operators
7. Computing a rigid transformation (registration) that best aligns different shapes

Many popular algorithms applied to these problems are based on using the shortest distance measure between sets. This measure does not take local surface orientation into account and is asymmetric, and hence, suboptimal. One commonly used algorithm for computing correspondences between registered shapes is based upon closest

point projections for all points from A to B or B to A, or equivalently, the normal projection from B to A or A to B [25]. To be ideal, we argue that a correspondence algorithm should be symmetric and return the same result regardless of order of operation.

Hausdorff distance [53] is a common measure for computing the global discrepancy between two shapes. It is also based on closest projection (point-to-shape minimal distance). Because it uses this closest-point computation, the Hausdorff distance between two shapes A and B must be computed twice, once from A to B, and then from B to A. The maximum of those two measures is then used. We argue that a better discrepancy measure is one that does not require this double computation and takes into account curve or surface orientation. Hausdorff distance is further discussed in Chapter 6.

In an attempt to improve upon these techniques, the present thesis explores the theoretical and algorithmic aspects of a novel difference measure, based on fitting a ball (disk in 2D) such that it has at least one tangential contact with each shape (curve in 2D or surface in 3D) and the ball interior is disjoint from both shapes (Fig. 1 center). This construction is both symmetric and orientation sensitive.

We propose a set of ball-based operators and discuss their properties, implementations, and applications. We divide the group of ball-based operations into unary and binary as follows:

Unary operators include:

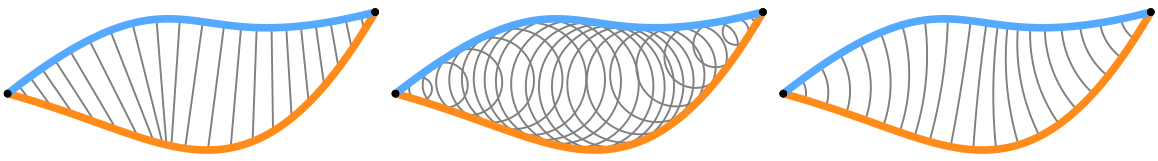


Figure 1: Left: Closest Point projections from the orange curve to the blue curve. Center: Balls tangent to both curves. Right: Circular Arc mappings corresponding to the *tangent-balls* (center).

- identifying details (salient features, constrictions, sharp features)
- smoothing shapes by removing such details, replacing them by fillets and roundings
- segmentation (recognition, abstract modelization via centerline and radius variation) of tubular structures

Binary operators include:

- measuring the local discrepancy between two shapes
- computing the average of two shapes
- computing point-to-point correspondence between two shapes
- computing circular trajectories between corresponding points that meet both shapes at right angles
- using these trajectories to support smooth morphing (inbetweening)
- using a curve morph to construct surfaces that interpolate between contours on consecutive slices

The technical contributions of this thesis are focused on the implementation of tangent-ball operators and their application to shape processing tasks. We show specific applications in the areas of animation and computer-aided medical analysis. These algorithms are simple to implement, mathematically elegant, and fast to execute.

1.1 Target Applications

Morphing is a fundamental tool in animation design where in-between frames are produced from a set of key-frames [21]. Artists would like to have control over correspondence as well as certain feature point trajectories. Once matches and control

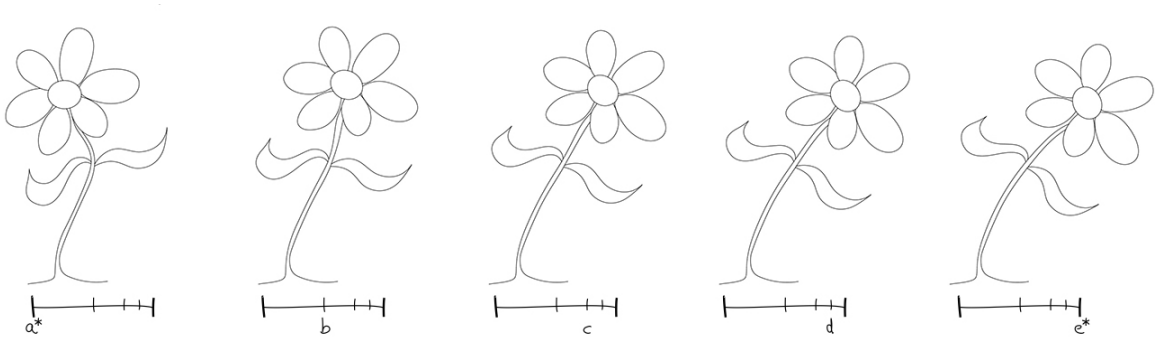


Figure 2: Morphing is a fundamental tool for creating inbetweens (b, c, d) from artist-produced keyframes (a, e) in an animation pipeline.

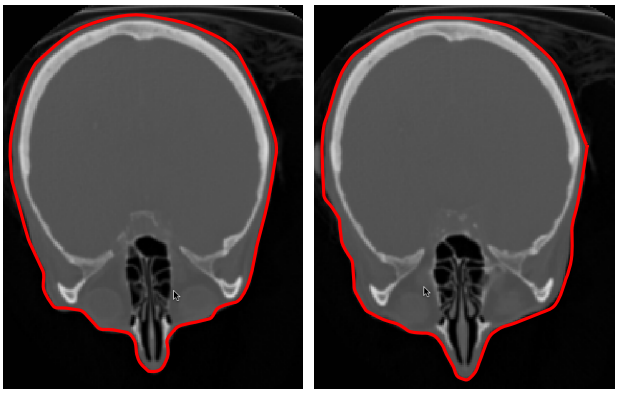


Figure 3: Two consecutive slices of a CT scan of a human head. The manually segmented closed contours are shown in red.

trajectories have been specified, either automatically or by the artist, the problem can be broken into a series of single-stroke inbetweens (morphing animations) which can sometimes be produced automatically, especially when the keyframes are very close together, as in Figure 2. In addition to key frame interpolation, morphing is also fundamental to producing models from slice contours extracted from medical image data, like the slices shown in Figure 3. We present here a morph which can accomplish these tasks, and we compare it to other morphs in the same context.

The blending (rounding and filleting) of sharp features is a necessary operation in the manufacturing of goods and a vital tool in CAD software. Our contribution to this application is based on the blending of one shape relative to a second "control" shape. This control shape defines the region to be blended as well as the local radius

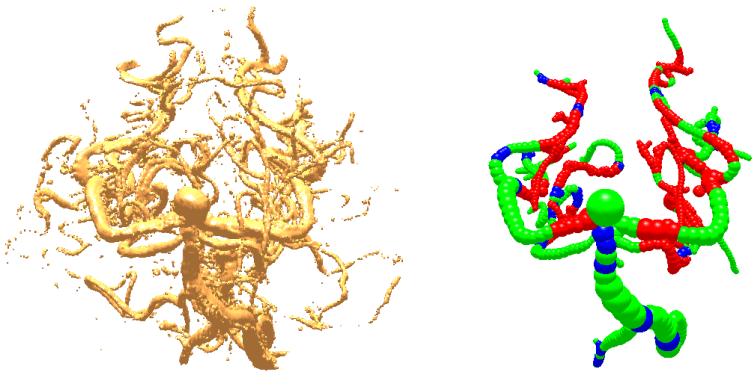


Figure 4: Given raw medical scan data (left), image segmentation and vessel extraction are important for producing clean geometry (right).

of the blend. The sharp features are replaced with smooth surfaces defined by the boundary of the sweep of a ball that rolls between the surfaces while staying in tangential contact with each.

There are numerous algorithms for the segmentation of medical images, both in 2D and 3D [63]. Our contribution in this area is a user-guided vessel segmentation algorithm which operates at interactive speeds. Other segmentation algorithms require between minutes and hours to compute a segmentation, which may need to be adjusted, at the cost of additional execution time. Our system allows for user guiding and editing of the result by culling unwanted branches and extending branches missed by the automatic first pass, an example of which is shown in Figure 4.

1.2 Structure of the thesis

Chapter 2 introduces the key properties of tangent balls and reviews previous ball-based techniques. Chapter 3 discusses the formulation of *ball-compatibility*, a fundamental property of pairs of curves and shapes on which this thesis is based. Chapter 4 describes *relative blending*, our tool for identifying and smoothing incompatible features. Chapter 5 introduces the *ball-map*; its definition and implementations for shape correspondence. Next, in Chapter 6, we discuss shape discrepancy and our ball-based

measure. Utilizing the *ball-map* correspondence, Chapter 7 defines the *ball-morph* - an animation between *ball-compatible* shapes. Chapter 8 compares the *ball-morph* to other morphs using a collection of measures. For dealing with shapes that are not *ball-compatible*, Chapter 9 introduces an extended *ball-morph*. Chapter 10 discusses how tangent balls can be used in the segmentation of tubular structures in 2D and 3D images. Then, Chapter 11 discusses techniques for computing a smooth tubular surface that bounds the tubes. Finally, Chapter 12 describes which techniques can be applied to medical image processing, while Chapter 13 discusses the relevant applications to the animation industry.

CHAPTER II

TANGENT BALLS

The ability of an algorithm to accomplish a shape processing task is limited to its ability to perceive and understand the input. The algorithm designer supplies this knowledge as a fixed set of rules meant to account for a range of variables in the input. These rules are often expressed in terms of geometric primitives (points, lines, planes, directions) of operations (fitting, intersection) and of measures (distance, proximity). We focus on techniques where the primitive is a ball, the operation is fitting it to achieve several tangential contacts, and the measure is the radius of the ball. Using a ball in such a manner is easy to understand and natural for an algorithm designer, and ultimately a user.

2.1 Ball Properties

We define it in n-dimensions as a set $\mathbb{B}(\mathbf{c}, r)$ of all points with distance less than or equal to a radius r from a center point $\mathbf{c}$. The equation of a closed ball is

$$\mathbb{B}(\mathbf{c}, r) = \{\mathbf{p} : \overrightarrow{\mathbf{cp}}^2 \leq r^2\} \tag{1}$$

where $\overrightarrow{\mathbf{cp}}$ is the vector such that $\mathbf{p} = \mathbf{c} + \overrightarrow{\mathbf{cp}}$ and $\overrightarrow{\mathbf{cp}}^2$ stands for $\overrightarrow{\mathbf{cp}} \cdot \overrightarrow{\mathbf{cp}}$ where $\cdot$ is the dot product. In 2D, a ball is a disk and in 3D it is a solid ball. The boundary of a ball in 2D is a circle, and in 3D, a sphere.

In order to compute whether a point $\mathbf{p}$ lies within a ball of center $\mathbf{c}$ and radius r, the following inequality is tested:

$$\overrightarrow{\mathbf{cp}}^2 \leq r^2. \tag{2}$$

The signed distance d between the boundary of a ball and a point $\mathbf{p}$ is computed as

$$d = \sqrt{\overrightarrow{\mathbf{cp}}^2} - r \tag{3}$$

where a negative value of d indicates that $\mathbf{p}$ is within the ball.

Two balls, $\mathbb{B}_1(\mathbf{c}_1, r_1)$ and $\mathbb{B}_2(\mathbf{c}_2, r_2)$, are in tangential contact when the following equality is true:

$$\overrightarrow{\mathbf{c}_1\mathbf{c}_2}^2 = (r_1 + r_2)^2. \tag{4}$$

Any object that intersects the boundary but not the interior of a ball is tangent to that ball at the contact points. If you hold a ball in your hand, every point on your hand that is touching the ball is tangent to the ball. If a ball is tangent to multiple objects, the center of the ball is equidistant from all contact points with distance equal to the radius. Hence, it is computationally easier to test or enforce tangential contact with a ball than with more complex shapes.

A ball $\mathbb{B}(\mathbf{c}, r)$ is tangent to a smooth shape Q at point $\mathbf{p}$ with surface normal $\widehat{\mathbf{N}}_Q(\mathbf{p})$ when the following equality is true:

$$\mathbf{c} = \mathbf{p} + r\widehat{\mathbf{N}}_Q(\mathbf{p}). \tag{5}$$

A shape is smooth when there is a well defined unique normal at each point on its boundary and when the field of normals is continuous.

2.2 Additional Ball-based Tools

Additional primitive tools which are important to several tangent-ball based algorithms are described here, in the form of geometric constructions.

2.2.1 Circle through 3 planar points

Given a set of 3 non-colinear points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$, the center $\mathbf{c}$ of the ball $\mathbb{B}(\mathbf{c}, r)$ that is tangent to all three points is found by computing the intersection of the perpendicular bisectors of $\overrightarrow{\mathbf{p}_1\mathbf{p}_2}$ and $\overrightarrow{\mathbf{p}_2\mathbf{p}_3}$:

$$\mathbf{m}_1 = (\mathbf{p}_1 + \mathbf{p}_2)/2 \tag{6}$$

$$\mathbf{m}_2 = (\mathbf{p}_2 + \mathbf{p}_3)/2 \tag{7}$$

$$\mathbf{c} = intersect(\mathbf{m}_1, R(\overrightarrow{\mathbf{p}_1\mathbf{p}_2}), \mathbf{m}_2, R(\overrightarrow{\mathbf{p}_2\mathbf{p}_3})) \tag{8}$$

$$r = |\mathbf{c} - \mathbf{p}_1| \tag{9}$$

where $R(\overrightarrow{\mathbf{v}})$ returns a vector orthogonal to $\overrightarrow{\mathbf{v}}$ and $intersect(\mathbf{s}_1, \overrightarrow{\mathbf{v}}_1, \mathbf{s}_2, \overrightarrow{\mathbf{v}}_2)$ returns the point intersection between two lines each defined by a point $\mathbf{s}$ and direction vector $\overrightarrow{\mathbf{v}}$.

2.2.2 Sphere through four points

Given a set of 4 nonplanar and non-colinear points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4$, computing a ball $\mathbb{B}(\mathbf{c}, r)$ that is tangent to all 4 points is similar to the 3 points method, such that the center $\mathbf{c}$ is found by computing the intersection point of the bisecting planes of $\overrightarrow{\mathbf{p}_1\mathbf{p}_2}$, $\overrightarrow{\mathbf{p}_2\mathbf{p}_3}$ and $\overrightarrow{\mathbf{p}_3\mathbf{p}_4}$:

$$\mathbf{m}_1 = (\mathbf{p}_1 + \mathbf{p}_2)/2 \tag{10}$$

$$\mathbf{m}_2 = (\mathbf{p}_2 + \mathbf{p}_3)/2 \tag{11}$$

$$\mathbf{m}_3 = (\mathbf{p}_3 + \mathbf{p}_4)/2 \tag{12}$$

$$\mathbf{c} = intersect(\mathbf{m}_1, \frac{\overrightarrow{\mathbf{p}_1\mathbf{p}_2}}{|\overrightarrow{\mathbf{p}_1\mathbf{p}_2}|}, \mathbf{m}_2, \frac{\overrightarrow{\mathbf{p}_2\mathbf{p}_3}}{|\overrightarrow{\mathbf{p}_2\mathbf{p}_3}|}, \mathbf{m}_3, \frac{\overrightarrow{\mathbf{p}_3\mathbf{p}_4}}{|\overrightarrow{\mathbf{p}_3\mathbf{p}_4}|}) \tag{13}$$

$$r = |\mathbf{c} - \mathbf{p}_1| \tag{14}$$

where $intersect(\mathbf{s}_1, \widehat{\mathbf{N}}_1, \mathbf{s}_2, \widehat{\mathbf{N}_2}, \mathbf{s}_3, \widehat{\mathbf{N}_3})$ computes the intersection point of 3 planes defined by a point $\mathbf{s}$ and a normal vector $\widehat{\mathbf{N}}$.

2.2.3 Circle Inversion

A circle inversion is a transformation of the Euclidean plane that transforms points across the boundary of a circle. Given a point $\mathbf{p}$ and a reference circle $\mathbb{B}(\mathbf{c}, r)$, the transformed point $\mathbf{q}$ satisfies the equation

$$|\overrightarrow{\mathbf{cp}}| \cdot |\overrightarrow{\mathbf{cq}}| = r^2 \tag{15}$$

which is computed as

$$\mathbf{q} = \mathbf{c} + \frac{\overrightarrow{\mathbf{cp}} \cdot r^2}{\overrightarrow{\mathbf{cp}}^2} \tag{16}$$

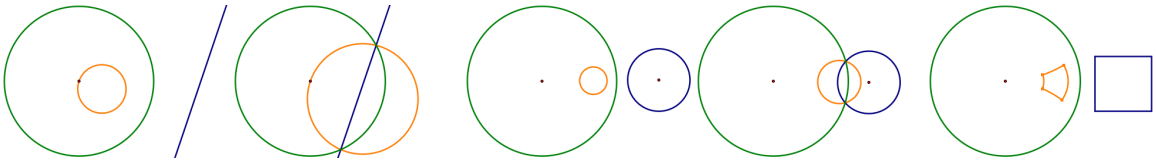


Figure 5: Several examples of circle inversion where the primitive in blue is the inverse of the primitive in orange across the green circle.

Circles on the interior of the ball that do not pass through **c** are transformed to circles on the exterior, and vice-versa (Fig. 2.2.3 (c)). If a circle on the interior does pass through **c** then it is transformed to a line (infinite radius circle) on the exterior since **c** transforms to infinity (Fig. 2.2.3 (a)).

This inverted geometry can make difficult problems more tractable once an inversion is applied since angles between lines and curves are preserved through inversion, as shown in Figure 2.2.3 (e), where the four corners of the inverted square are right angles. The concept of circle inversion can also be generalized to higher dimensions.

2.3 Tangent Balls in Prior Art

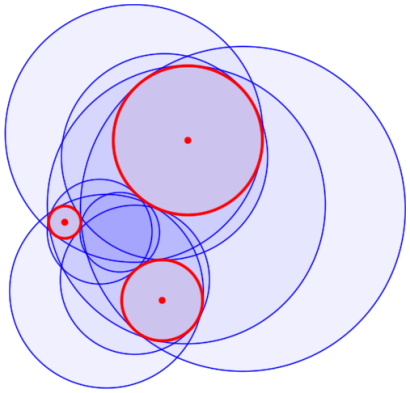


Figure 6: There are up to 8 solution tangent circles (blue) to any 3 given input circles (red).

2.3.1 Apollonius' Circles and Spheres

Now consider the computation of a ball that is tangent to several given balls. In 2D, this problem is a special case of a famous problem addressed more than 2000 years ago. The construction of tangent circles was first formally studied by Apollonius around 200 B.C. The problem he proposed, known as "Apollonius' Problem", is: given three objects, each of which may be a point, line or circle, construct a circle that is tangent to each. All possible cases were solved by construction between Apollonius and Euclid except for the case of three circles. A solution to this final case of Apollonius' problem was not discovered until 1816 by J.D. Gergonne [50]. The important tool that was needed in order to solve this problem was an understanding of circle inversion [67]. His solution is a pure construction and can be done using a compass and straight edge. This construction yields the 8 solution tangent circles given three circles in general positions, as shown in Fig. 6.

The Gergonne construction is outlined below. In order to obtain all possible solutions for a given configuration, all three input circles must have different radii.

1. For each pair of circles, construct the radical axis, yielding 3 radical axis lines
2. Construct the radical center, which is the intersection of all 3 radical axes
3. For each pair of circles, construct the 2 dilation points, yielding 6 total dilation points
4. Construct 4 dilation lines that each intersect 3 dilation points
5. Choose 1 dilation line L, compute its inversion pole (point) for each of the 3 circles
6. For each circle, construct the line through the radical center and its inversion pole, yielding 2 points of intersection **a** and **b**, where **a** is the intersection point nearest the radical center

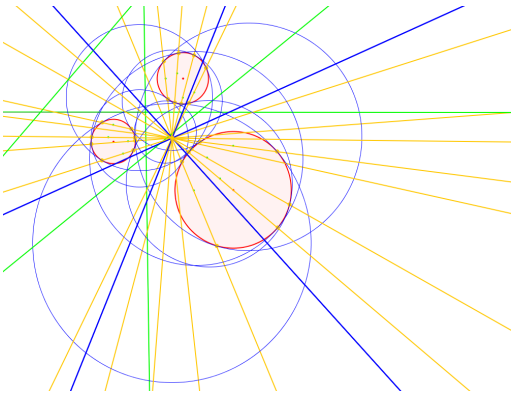


Figure 7: The Gergonne construction for computing the Apollonius circles.

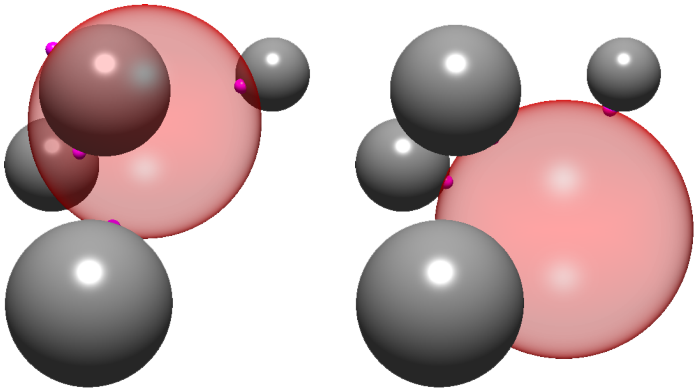


Figure 8: There are up to 16 solution tangent spheres (two shown here in red) to any 4 given input spheres (gray). Points of tangency are shown in purple.

7. Identify the circle C of the 3 circles that is alone on one side of the current dilation line L

8. Construct a solution circle through point **a** of circle C and points **b** from the other two circles

9. Construct a solution circle through point **b** of circle C and points **a** from the other two circles

10. Repeat steps 5-9 for the remaining 3 dilation lines

We have extended this construction to 3D. The construction requires the following modifications to the above steps:

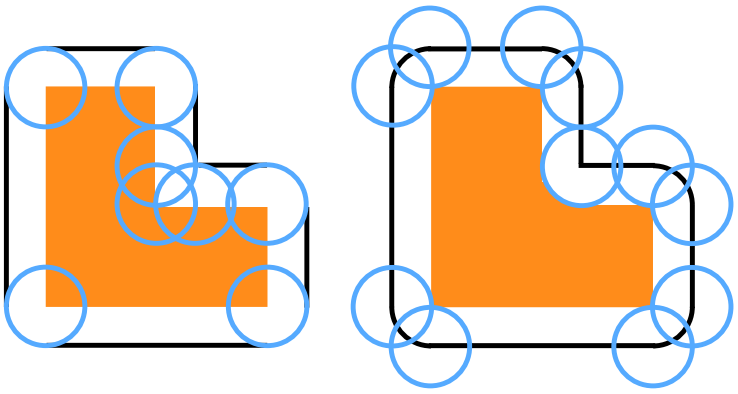


Figure 9: Left: The shape (orange) is grown by a constant radius, denoted by the blue circles, to obtain the black shape. Right: A constant-radius shrink operation is performed on the grown shape in order to produce a filleted version of the original shape (orange). Notice the fillet at the concave vertex.

1. 12 dilation points are constructed instead of 6 in 2D

2. 8 combinations of 4 of the 12 dilation points are coplanar and define 8 dilation planes, as opposed to 4 dilation lines in 2D

3. There are 8 radical axis planes instead of 4 radical axis lines, and they all intersect at a single point, the radical center

4. In the final step, spheres through 4 points are constructed instead of circles through 3 points

This produces a maximum of 16 tangent spheres to any given 4 input spheres. It should be noted that 16 solutions do not always exist just as 8 don't always exist in 2D. Also note that we have no proof that our construction always yields all possible solutions for a given configuration. Two example solution spheres are shown in red in Fig. 8.

2.3.2 Rounding and Filleting

Tangent balls have been used to compute rounds and fillets in solid models [96]. These constant radius blends of solids are defined in terms of growing and shrinking

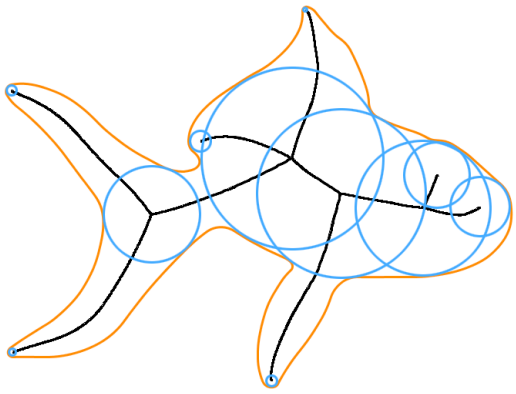


Figure 10: The medial axis (black) defines a set of points that are equidistant to two or more points on a curve (orange). The medial axis transform associates a radius at each medial axis point, defining a set of maximal balls (a subset are shown in blue), the boundary of the union of which fully define the original curve.

operations as morphological opening and closing [76] with a ball of constant radius r. Consider a shape S. We use respectively $!S$, iS, eS, bS, and cS, to denote the complement, interior, exterior, boundary, and closure of set S. The r-filleting $F_r(S)$ of a set S is defined as $(S^r)_r$, where S^r is S grown by r (the set of all points for which the distance to S does not exceed r) and where $S_r = !((!S)^r)$, yielding $F_r(S) = (S^r)_r = !((!(S^r))^r)$. $F_r(S)$ is the space not reachable by an open ball of radius r that remains disjoint from S. Figure 9-right (orange) shows the result of an r-filleting on the solid in Figure 9-left (orange). Similarly, the r-rounding $R_r(S)$ of a solid S is defined as $(S_r)^r$ and is the set reachable by a closed ball of radius r in S.

2.3.3 Maximal Balls

Given a closed and regularized [114] set S, following [108], we say that a ball in S is *maximal* if it is not contained in any other ball in S. The *medial axis* of S is defined as the set of all points having more than one closest point to the boundary bS. The medial axis axis M of S in the interior iS is the closure of the centers of all *maximal-balls* in S. The medial axis is a set of curves (and/or points) for shapes in $\mathbb{R}^2$ and a set of surfaces (and/or curves, points) in $\mathbb{R}^3$. The *medial axis transform* (MAT) of S is the interior medial axis M of S with an associated radius (distance to

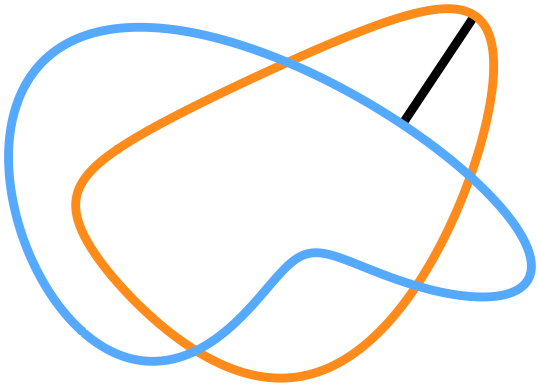


Figure 11: The Hausdorff distance is a common discrepancy measure. In this example, it is represented by the black line.

bS) for each sample $\mathbf{m}$ on M. The possibly infinite union of the balls of the MAT of S, is equal to S:

$$S = \bigcup_{\mathbf{m} \in M} \mathbb{B}(\mathbf{m}, r(\mathbf{m})). \tag{17}$$

In [20], state-of-the-art algorithms are given for computing the medial axis M of a planar curve, as well as for recovering the original curve as the boundary of the union of maximal balls described by the medial axis transform. Figure 10 shows an example of the interior medial axis of a closed curve, and also shows several of the maximal balls that define the medial axis transform.

The term *local feature size* at a point $\mathbf{s}$ on the boundary of S defines the closest distance of $\mathbf{s}$ to the medial axis of S. Note that this includes the interior and exterior portions of the medial axis. We use the term *minimum feature size* (or *reach* [41]) of the set S as the smallest of all local feature sizes. Minimum feature size may also be defined as the largest r for which $F_r(S) = S$ and $R_r(S) = S$.

2.3.4 Hausdorff Distance

The Hausdorff distance [53] is related to the distance between two sets and is often used as a discrepancy measure. $H(A, B)$ is the Hausdorff distance between A and B and is the smallest r for which $A \subset B^r$ and $B \subset A^r$.

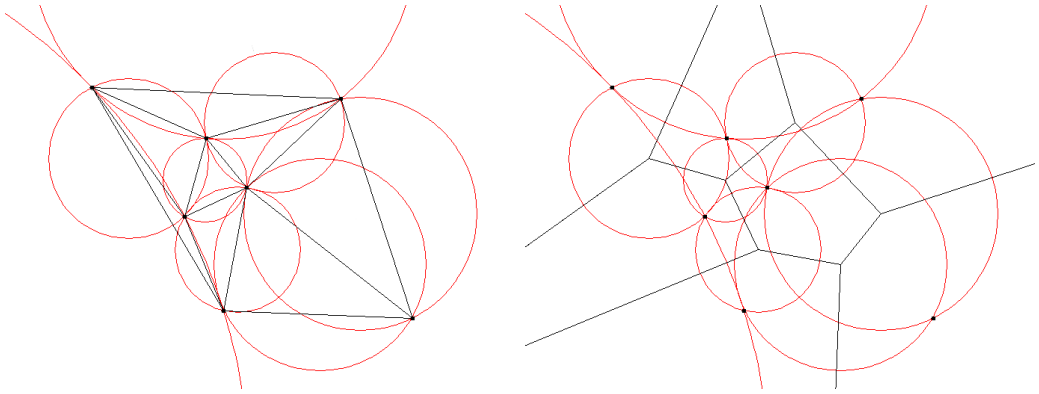


Figure 12: Left: The Delaunay triangulation of the given points and the associated circumscribing circles for each triangle. Right: The Voronoi tesselation, or medial axis, of the given set of points.

The Hausdorff distance is based upon the point-to-shape minimal distance measure. The distance between a point $\mathbf{a}$ and a shape B is defined simply as $d(\mathbf{a}, B) = \min_{\mathbf{b}\in B}(dist(\mathbf{a}, \mathbf{b}))$. This distance identifies one or several points $\mathbf{b} \in B$ that are as close to $\mathbf{a}$ as possible. Extending this definition to two shapes, the Hausdorff distance is equivalently defined as

$$H(A, B) = \max(\max_{\mathbf{a}\in A}(d(\mathbf{a}, B)), \max_{\mathbf{b}\in B}(d(\mathbf{b}, A))). \tag{18}$$

An example of this distance is shown in Fig. 11. If the curves A and B are thought of as sidewalks and a different person is constrained to walk on each, the Hausdorff distance defines that maximum distance apart the two people could be if one is in pursuit of the other.

2.3.5 Delaunay and Voronoi

If S is a set of discrete points, then the medial axis of S is equal to the union of the Voronoi edges of those points, an example of which is shown in Fig. 12 (right). Every point on an edge of the Voronoi tessellation corresponds to the center of a circle that is tangent to 2 points in S and contains no other points of S. The vertices of the Voronoi tessellation correspond to the centers of maximum empty circles that are tangent to 3 or more points in S and contain no other points of S. The dual of this

tessellation is the Delaunay triangulation, as shown in Fig. 12 (left). The circumcircle of a triangle in a Delaunay triangulation does not contain any other points from the set S, and the center of the circumcircle defines a vertex in the Voronoi tessellation.

There are numerous algorithms for computing Delaunay triangulations [115] with computational complexities in 2D ranging from $O(n^4)$ to $O(n \log n)$ [44].

If the input S is a set of curves instead of points, then the curves of the medial axis are Voronoi curves and have the same properties as when S is a set of points. Likewise, if the input is a set of surfaces, the surfaces of the medial axis are Voronoi surfaces.

CHAPTER III

BALL COMPATIBILITY

3.1 Compatibility

In general, we use the term *compatibility* and say that two given sets A and B are compatible when there exists a function $m_c : A \to B$ that is a homeomorphism (i.e. is a bijection, is continuous, and has a continuous inverse m_c^{-1}). In this section, we define *ball-compatibility* (abbreviated *b-compatibility*) between curves, and discuss how it relates to the Hausdorff distance, and also how it compares with the closest-point compatibility, or *c-compatibility*. We illustrate compatibility in terms of curves in $\mathbb{R}^2$ for clarity. These definitions apply to surfaces in $\mathbb{R}^3$ as well.

A curve is *simple* if it is planar, connected, free from self-intersections, and if it is topologically closed (contains its endpoints, if any). A simple curve is either a *loop* (no end-points) or a *stroke* (two endpoints). We consider two simple curves, A and B, that are either both closed or both strokes with identical endpoints.

We contrast ball compatibility, or *b-compatibility*, with the following definition of *c-compatibility*.

Definition 1 (c-compatibility) *A and B are said to be* c-compatible *if for every point $\mathbf{a} \in A$ there exists exactly one point $\mathbf{b} \in B$ such that there exists an offset $d(\mathbf{b})$ such that $\mathbf{a} = \mathbf{b} + \widehat{\mathbf{N}}_B(\mathbf{b})d(\mathbf{b})$ and for every point $\mathbf{b} \in B$ there exists exactly one point*

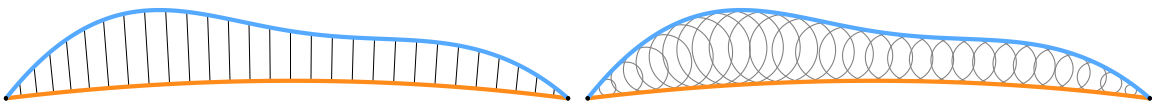


Figure 13: Left: The blue curve is expressed as the normal offset of the orange curve. Right: The blue curve is expressed as the ball-offset of the orange curve (and vice-versa).

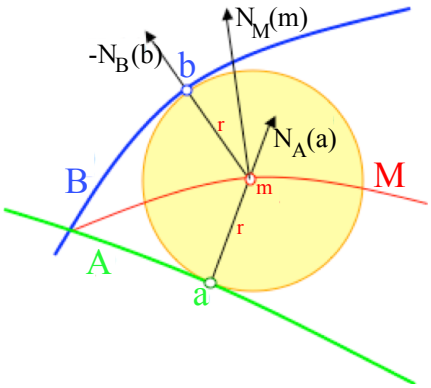


Figure 14: Reconstruction of the *ball offset*

$\mathbf{a} \in A$ *such that* $\exists d(\mathbf{a})$ *such that* $\mathbf{b} = \mathbf{a} + \widehat{\mathbf{N}}_A(\mathbf{a})d(\mathbf{a})$.

Two curves are *c-compatible* when each one can be expressed as the normal offset of the other, i.e., as a continuous height-field orthogonal to the other curve at all points (Fig. 13-left). A more precise definition of *c-compatibility* is discussed in [25].

Definition 2 (b-compatibility) *A and B are said to be* b-compatible *if for every point $\mathbf{a} \in A$ there exists a ball $\mathbb{B}(\mathbf{c}, r)$ that is tangent to A only at $\mathbf{a}$ and also tangent to B at exactly one point $\mathbf{b} \in B$.*

When two curves are *b-compatible*, each can be expressed as the ball-offset of the other, i.e., as a portion of the envelope swept by a variable radius disk as it rolls on the other curve, assuming neither curve is self-intersecting (Fig. 13-right).

3.1.1 Ball offset

When A and B are *b-compatible*, B is the *ball offset* of A and hence can be represented by A and a height field $r(A)$. In practice, to compute the *ball offset* representation of a shape B, we sample A and for each sample, $\mathbf{a}$, record the associated r-value, which is the radius of the ball that is tangent to A at $\mathbf{a}$. If necessary, we recover a continuous approximation of the height field using linear or higher order interpolation.

Given a point $\mathbf{a} \in A$ and the field value $r(\mathbf{a})$ at $\mathbf{a}$, we reconstruct the corresponding point $\mathbf{b} \in B$ as follows (Fig. 14). First, we compute the normal $\widehat{\mathbf{N}}_A(\mathbf{a})$ at $\mathbf{a}$ from the

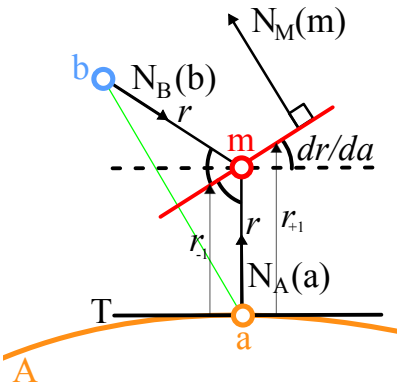


Figure 15: Reconstruction of the normal $\widehat{\mathbf{N}}_M(\mathbf{m})$ for use in the *ball offset* construction.

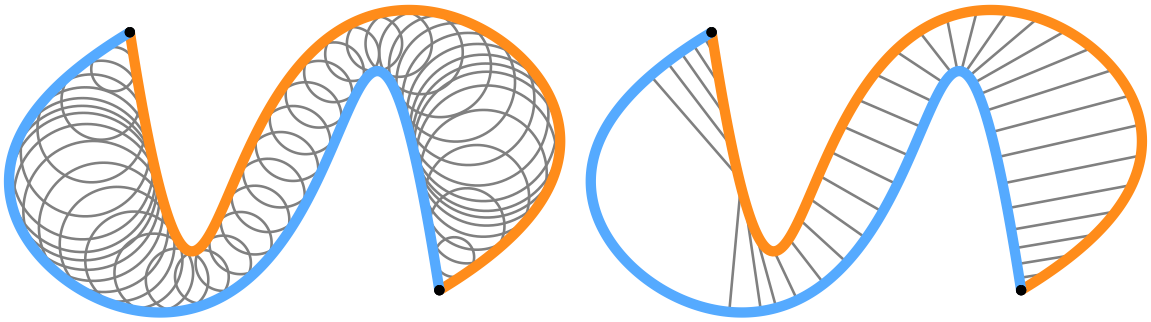


Figure 16: Two curves that are *b-compatible* (left), but not *c-compatible* (right).

representation of A and then construct the corresponding point $\mathbf{m} = \mathbf{a} + r(\mathbf{a})\widehat{\mathbf{N}}_A(\mathbf{a})$, which lies on the medial axis M between A and B. Then, we must compute the normal $\widehat{\mathbf{N}}_M(\mathbf{m})$. We compute this normal by measuring the local derivative of r with respect to the arc-length of A as follows. As shown in Figure 15, we compute a line T tangent to A at $\mathbf{a}$ and use it as the base of a height-field of the local values of r. Using this local estimate of M, we are able to compute $\widehat{\mathbf{N}}_M(\mathbf{m})$ by rotating $\widehat{\mathbf{N}}_A(\mathbf{a})$ by the angle $\mathbf{tan}^{-1}((r_{+1} - r_{-1})/dist(\mathbf{a_{+1}}, \mathbf{a_{-1}}))$ where $_{+1}$ and $_{-1}$ denote next and previous, respectively. Finally, we compute $\mathbf{b} = \mathbf{a} + 2(\overrightarrow{\mathbf{am}} \cdot \widehat{\mathbf{N}}_M(\mathbf{m}))\widehat{\mathbf{N}}_M(\mathbf{m})$ using the property that $\triangle\mathbf{amb}$ is an isosceles triangle such that $\overrightarrow{\mathbf{ab}}$ is parallel to $\widehat{\mathbf{N}}_M(\mathbf{m})$ and $|\overrightarrow{\mathbf{ab}}| = 2\widehat{\mathbf{N}}_M(\mathbf{m}) \cdot \overrightarrow{\mathbf{am}}$.

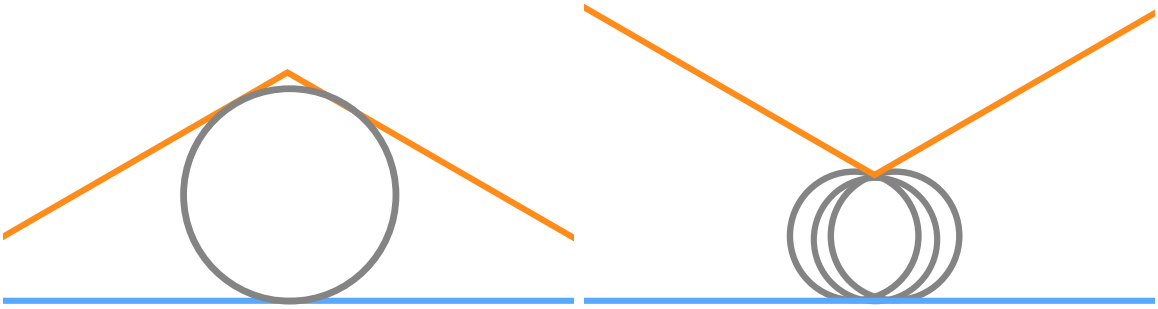


Figure 17: If the curves are not smooth, they are not *b-compatible*. Left: if a sharp concave feature exists, a maximal ball cannot reach the sharp vertex. Right: if a sharp convex feature exists, there are several maximal balls which are tangent to the sharp vertex. For these reasons, a homemorphism using maximal balls cannot exist in either case.

3.1.2 Bounds of b-compatibility

Chazal et al. [24] show that if two curves A and B are smooth loops and their Hausdorff distance H is less than the minimum feature size (mfs) of both A and B, then A and B are *b-compatible*. Note that this is a sufficient, but not necessary condition.

In contrast, Chazal et al. [25] show that two curves are *c-compatible* when their minimum feature size f and their Hausdorff distance H satisfy the following relation: $H < (2 - \sqrt{2})mfs$.

Compare this bound with the bound on *b-compatibility*: $H < mfs$. Observe that H is proportional to the discrepancy between two shapes and that $2 - \sqrt{2} \approx 0.58 < 1$. Thus, the *b-compatibility* condition is less constraining: Pairs of shapes may pass the *b-compatibility* condition, while they do not pass the *c-compatibility* condition (See Figure 16 for example).

The property of *b-compatibility* is only possible when the curves are smooth, i.e. at least C^1 continuous. Fig. 17 shows the problem when one of the curves contains a sharp feature. If one of the curves contains a concave feature (with respect to the other curve), there does not exist a maximal ball that is tangent to all points. Fig. 17

(left) shows the largest possible maximal ball for the curves. If one of the curves contains a convex feature (Fig, 17 (right)), there exists an infinite set of maximal balls that are tangent to the sharp feature. Because this vertex is tangent to multiple maximal balls, the other curve can no longer be represented as a ball-offset, because more than one ball radius would be required for the sharp point.

3.2 Testing for b-compatibility

We consider 3 possible types of curves A and B in the plane for testing *b-compatibility*:

1. Open curves with shared endpoints that meet at angles less than π (Fig. 3.2-left)
2. Closed curves such that $iA \cap iB$ and $eA \cap eB$ are each a non-empty connected set (Fig. 3.2-center)
3. Closed curves such that $A \cap B = \emptyset$ and $A \cap iB = A$ or $B \cap iA = B$ (Fig. 3.2-right)

Note that for a closed curve A, iA denotes the open region bound by A. Also note that type 2 can be decomposed into type 1 by cutting A and B at their intersection points.

There are several ways to test for *b-compatibility* given two curves A and B in the plane. Recall that *b-compatibility* is defined in terms of maximal balls that are tangent to both A and B. As discussed in Chapter 2, the closure of the centers of maximal balls inscribed in a set S defines the internal medial axis of S. Following this definition, *b-compatibility* can be discussed in terms of the internal medial axis M of the *moat* $X(A, B)$.

Definition 3 (moat)*: If A and B are closed loops, then the moat $X(A, B)$ is the closed set $(A \cup iA)\Delta(B \cup iB) \cup A \cup B$, where $(A \cup iA)\Delta(B \cup iB)$ is the symmetric difference (XOR) of the closed regions bound by A and B. If A and B are open curves that share endpoints and define a closed loop $C = A \cup B$, then $X(A, B)$ is defined as the closed set $C \cup iC$.*

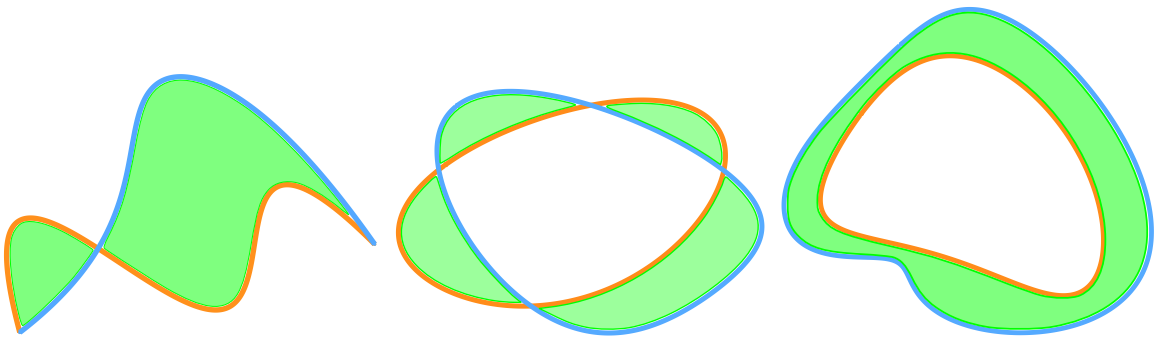


Figure 18: We consider three types of curves for *b-compatibility*: end-point aligned open curves (left), intersecting closed curves (center), and non-intersecting closed curves where one is bound by the other (right). The *moat* $X(A, B)$ is shaded in green for each.

Note that the definition includes $A \cup B$ so that X includes the points of intersection of A and B. Figure 3.2 shows $X(A, B)$ for the three possible types of input curves.

B-compatibility implies the following additional two conditions on the input curves A and B:

1. $i!X(A, B)$ has no more than 2 components

2. The internal medial axis of X is simply connected

If A and B belong to one of the three types and the internal medial axis M of $X(A, B)$ is known, then *b-compatibility* can be determined by examining the topology of M. If M is continuous and contains no bifurcations then A and B are *b-compatible*. Figure 19 shows an example where the internal medial axis contains a branch. The bifurcation point corresponds to a ball that is tangent to one of the curves at more than one point and the branch corresponds to maximal balls that are only tangent to the blue curve. Recall that in order for two curves to be *b-compatible*, the maximal balls must be tangent to each curve at exactly one point.

If the internal medial axis of X doesn't contain bifurcations, but does contain a sharp (C^0 continuous) feature, then A and B are defined as *quasi-b-compatible*. The maximal ball centered at a discontinuity touches one or both of the curves at an

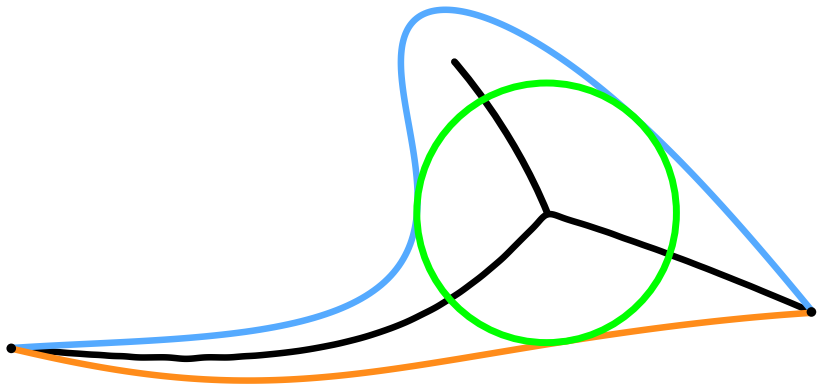


Figure 19: If the internal medial axis (black) of $X(A, B)$ contains bifurcations, then the curves are not *b-compatible*. The maximal ball centered at the bifurcation point touches the blue curve at 2 points.

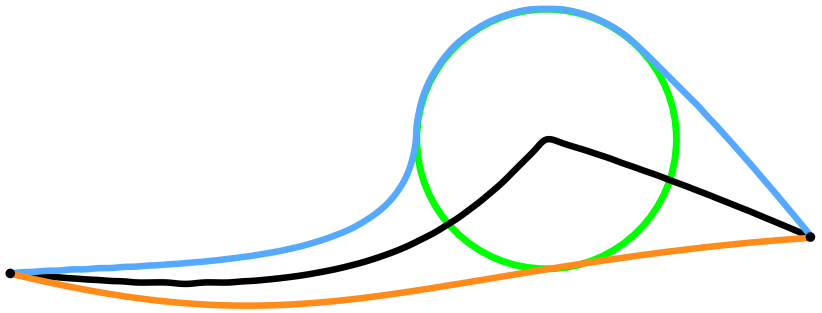


Figure 20: If the internal medial axis of X contains a discontinuity, then the curves are *quasi-b-compatible*, and contain a maximal ball (green) centered at each discontinuity that is tangent to an infinite set of points on one or both curves.

infinite set of points, as shown in Figure 20.

3.2.1 Algorithms

Piecewise-circular curves (PCCs) [97] are composed of circular-arc edges that meet tangentially (C^1 continuous) at vertices. We propose the following algorithm for testing *b-compatibility* between two PCCs A and B (Algorithm 1). We can use the Apollonius solution described in Chapter 2 in order to test if any ball exists that is tangent to three circular arcs, and if not, then A and B are *b-compatible*. For every combination of three circular arcs from $A \cup B$, the Apollonius solution circles

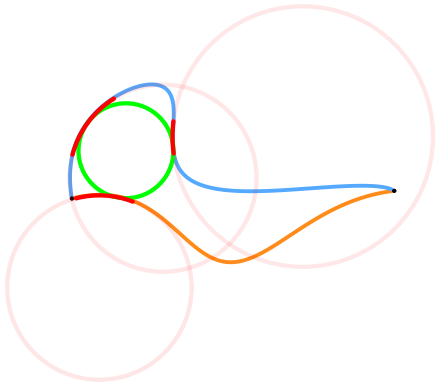


Figure 21: Given two piecewise-circular curves (orange, blue), then the Apollonius' circle solution (green) may be used to check for incompatible features.

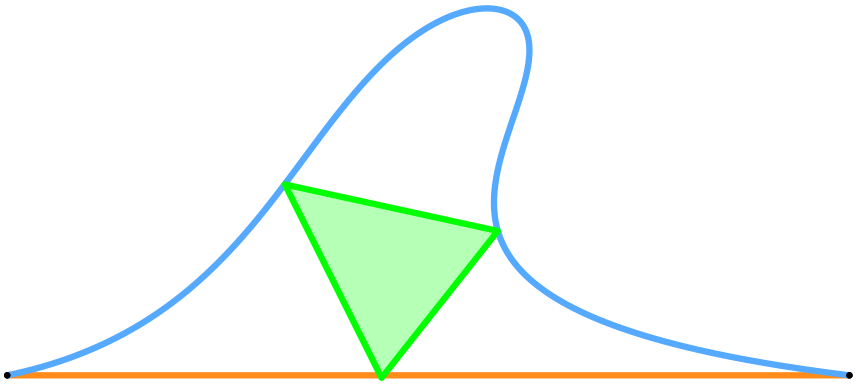


Figure 22: If a constrained Delaunay triangulation in X yields a triangle with no edges on either A or B, then A and B are not *b-compatible*.

are computed. Because the Apollonius solutions will be circles tangent to the three circles that define the three arcs, the solutions are then tested for tangency with the arcs. The remaining solutions that meet this criteria are then tested for intersection with A and B. If a solution is found that does not intersect A or B and is within X, then A and B are not *b-compatible*, as shown in Fig. 21.

As discussed in the previous section, our definition of *b-compatibility* doesn't allow for piecewise-linear curves or triangle meshes to be considered *b-compatible*. However, if the curves are sufficiently smooth and densely sampled, we propose a definition for the approximate *b-compatibility* of polygonal curves.

A and B are approximately *b-compatible* when there is a constrained Delaunay

Algorithm 1 Pseudo-code for testing *b-compatibility* between two piecewise-circular curves A and B using the greedy $O(n^4)$ algorithm. Returns *true* if A and B are *b-compatible*, *false* otherwise.

```
1: S ← A.arcs + B.arcs
2: for all arcs s of S do
3:    for all arcs t ≠ s of S do
4:       for all arcs u ≠ s, u ≠ t of S do
5:          T ← Apollonius(s.circle, t.circle, u.circle)
6:          for all circles c of T do
7:             if s.tangentTo(c) and t.tangentTo(c) and u.tangentTo(c) then
8:                valid ← true
9:                for all arcs v ≠ s, v ≠ t, v ≠ u of S do
10:                  if c.intersects(v) then
11:                     valid ← false
12:                  end if
13:               end for
14:               if valid then
15:                  return false
16:               end if
17:            end if
18:         end for
19:      end for
20:   end for
21: end for
22: return true
```

triangulation of the vertices of the moat $X(A, B)$ and for each Delaunay triangle, exactly one edge is an edge of A or B. Figure 3.2.1 shows an example of a Delaunay triangle that violates this definition. Such a Delaunay triangle corresponds to a circumscribing ball that is tangent to one of the curves at more than one point, thereby violating the definition of *b-compatibility*.

A faster algorithm for approximating the *b-compatibility* test on piecewise-linear curves can be found in Chapter 9.

CHAPTER IV

RELATIVE BLENDING

Solid models may be blended through filleting or rounding operations that typically replace the vicinity of concave or convex edges by blends that smoothly connect to the rest of the solid's boundary. Circular blends, which are popular in manufacturing, are each the subset of a canal surface that bounds the region swept by a ball of constant or varying radius as it rolls on the solid while maintaining two tangential contacts. We propose to use a second solid to control the radius variation. This new formulation supports global blending (simultaneous rounding and filleting) operations and yields a simple set-theoretic formulation of the relative blending $R_B(A)$ of a solid A given a control solid B. We propose user-interface options, describe practical implementations, and show results in 2 and 3 dimensions.

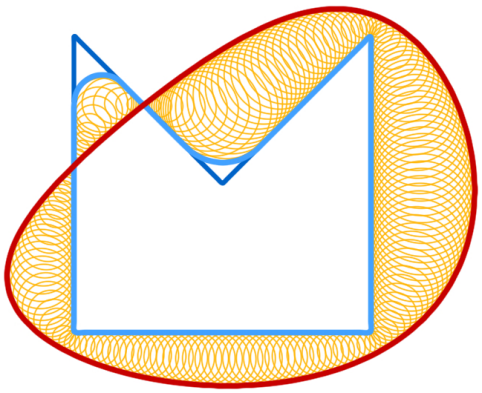


Figure 23: A simple example showing A (dark blue) and control shape B (red), with disks of O (orange) in the moat $X(A, B)$. The union of all disks in O defines the pad P. The final blended shape $R_B(A)$ is shown in light blue.

4.1 Introduction

Blending replaces portions of the boundary near the sharp features or constrictions of a solid model A by smooth surfaces (blends) that connect tangentially to the remaining part. Some blending operations are associated with one or several edges of a solid. Others produce smooth connections between non-intersecting portions of a solid or of two different solids. The formulation proposed here handles both types. In the absence of aesthetic or dynamic flow considerations, which may impose functional constraints on the nature of the blends, drafting and manufacturing practices often call for circular blends, which have circular cross-sections [57]. Such blends are part of the boundary of a canal surface [81], which is swept [14] by a ball as it rolls while maintaining two or more tangential contacts with the boundary of A. When the ball rolls in the interior of A, the removed material is called a rounding. When the ball rolls in the exterior of A, the added material is called a fillet. If the radius of the ball is constant the corresponding r-rounding and r-filleting operations have a set-theoretic formulation [96] as morphological opening and closing [105] with an r-ball. They may be applied to the entire solid or combined with Boolean operations to restrict their effect to a portion of the solid. In this chapter, we propose a set-theoretic formulation for variable-radius blending. The formulation of the proposed *relative blending* uses a "bounding" solid B to control the radius of the rolling ball locally. Intuitively, the rolling ball is required to stay in the symmetric difference, $A \Delta B$, and to be touching the boundary of A and B at all times, as shown in 2D by the disks in Fig. 23.

As such, our relative blending is a global operation that blends the entire solid. Hence the designer may not need to invoke specialized blending operations for corners where several blends meet. Because it offers a set theoretic formulation for the resulting set, *relative blending* may be incorporated within a CSG design paradigm [60] [92] and, if needed, combined with Boolean operations to localize the effect of blending.

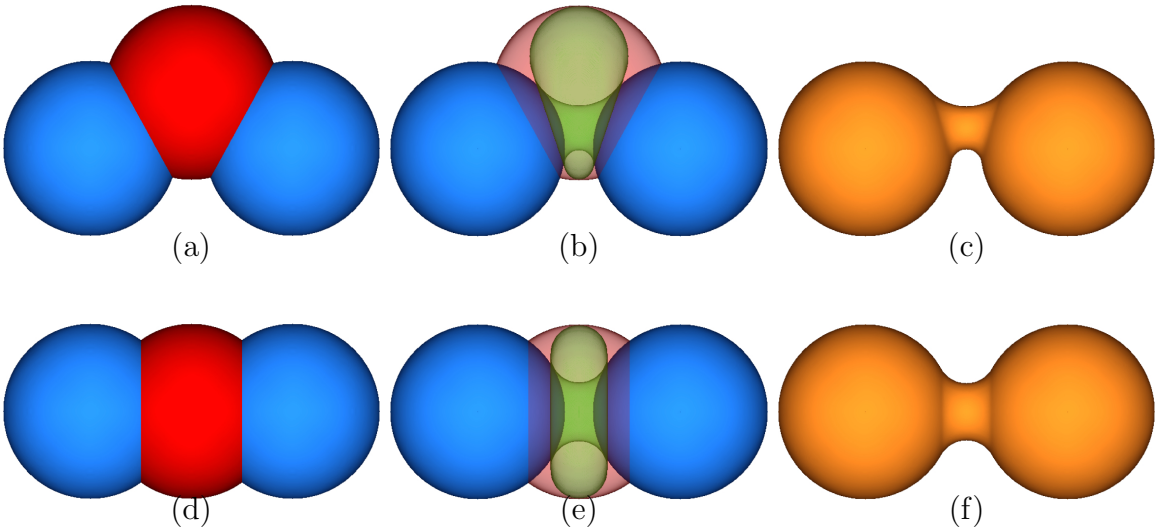


Figure 24: (a) An input shape A (blue) and a control shape B (red). (b) The Dupin cyclide (green) associated with the blend shown within B, and (c) the final rounded shape $R_B(A)$. (d) The same setup as (a) but with B directly in the center of A. (e) Constant radius Dupin cyclide (torus) shown within B, and the (f) final blending $R_B(A)$.

Note that when A is the Boolean combination of two balls and B is a third ball (Fig. 24), the blending surface is a portion of a Dupin cyclide [39]. Dupin cyclides have been successfully used to compute smooth approximations of individual blends. Hence, one may view the proposed approach as an extension of the Dupin cyclide construction to more complex shapes A and B.

Observe that the spine (curve traced by the center of the rolling ball) must lie on the medial axis (also called Voronoi) surface [84] of A (for rounding) or of its complement, $!A$ (for filleting). It has been suggested that variable-radius blends could be specified by defining such a spine, either by drawing an approximating space curve and snapping it to the medial axis surface [87] or by defining it as the intersection of the medial axis surface with a control surface C [58]. Another option is to draw one contact curve (also called a linkage curve or spring curve [18]) on A and compute the spine from it. The *relative blending* approach proposed here offers a different specification mechanism, which may prove more convenient for the designer. For example, the designer may start by setting B to be identical to A and then modifying

it locally through precise CSG operations or through less precise warping by pulling its boundary away from the boundary of A. Intuitively, pulling further increases the radius of the blend.

Finally, note that our relative blending may be used to perform both filleting and rounding in a single blending operation.

In conclusion, we suggest that the proposed *relative blending* paradigm offers a useful alternative to previously proposed approaches to the specification of variable radius blending, because it offers a simple set-theoretic formulation, operates globally on an entire solid, and simplifies the specification of complex blends.

To validate and demonstrate the proposed formulation, we have implemented it in two dimensions for cases where both A and B are bounded by PCCs (curves made of smoothly-connected line segments and circular arcs) [97] and in three dimensions by voxelizing [19] [71] A and B, computing the volumetric model of the relative blending of A with respect to B and generating the result as an iso-surface. Both implementations are straightforward. Note that the first one is exact, while the second one is an approximation, the accuracy of which may be controlled by adjusting the grid size.

The computation of the exact shape of these blends in 3D is tractable for simple situations where the blend is part of a Dupin cyclide, such as the example in Fig. 24. In general, however, the resulting canal surfaces are highly complex [28]. Hence, we advocate approximation or discretization, as was previously done for constant and variable radius blends [97] [28]. Because the combination of the voxelization process and its reverse (the iso-surfaces extraction) perform a resampling, they will typically alter A away from the blends. Hence, a more precise approach is to trace the variable-radius canal surfaces by "rolling" a variable radius ball that maintains two tangential contacts with the boundary of A and one with the boundary of B. The center of the rolling ball (i.e. the spine of the blend) follows an edge (seam [107]) of

the Voronoi (medial axis) surface complex of the moat X. For simplicity, throughout this chapter, we assume that A and B are closed-regularized (i.e. equal to the closure of their interior).

The remainder of this chapter is organized as follows. We first discuss prior art. Then we present the set theoretic formulation of relative blending. Then, we discuss two different implementations. Finally, we propose user interface options and mention applications.

4.2 Related Work

To put our contribution in perspective with a vast body of prior art on blending, we organize prior approaches into two categories: (1) the proposed *relative blending* approach is a ball-rolling technique, hence we contrast it with previously proposed constant-radius and variable-radius ball-rolling techniques. (2) a broad variety of other techniques for smoothing the sharp features of 2D shapes or 3D solids have been proposed. We briefly mention a few examples to provide a global context of our work and to point out similarities. Several articles include more comprehensive surveys of the topic [129] [119].

4.2.1 Ball rolling

In this subsection, we focus on blends generated by rolling a ball [100] of constant or varying radius [95] [118]. This is an important category, because often manufacturing and drafting practices require that the cross-section profile of a blend be circular [57].

In [96], Rossignac and Requicha formulate a global constant radius blending (rounding or filleting) of solids in terms of growing and shrinking operations as morphological opening and closing [76] with a ball of constant radius r, as discussed in Chapter 2. Recall that we use the term *blending* to refer to both filleting and rounding.

The advantage of this approach is that it offers a formal and unambiguous set-theoretic specification of the result of a blending operation applied to an entire solid, including corners where three or more blends meet. The specification is simple (indicate whether you wish to fillet or blend A and specify r) and the resulting solids are always well defined. The effect of such blending operations may be restricted to user-selected features by combining blending and Boolean operations. The relative blending solution proposed here preserves the advantage of constant radius blending by offering an unambiguous set theoretic specification for a global blending, yet it overcomes the constant-radius restriction.

A constant-radius blending operation replaces portions of the boundary of A with blending faces. Each blending face (simply called *blend* hereafter) is either a portion of a sphere of radius r that touches the boundary of A at three or more points or the subset of a constant-radius *canal-surfaces* [81] [55] that is the envelope of the region swept by a ball of radius r as it rolls on (i.e. remains in tangential contact with) two elements (faces, edges, vertices) of the boundary of A. Canal surfaces have a high algebraic degree, even when the two faces upon which the ball rolls lie on quadrics. To avoid dealing with high-degree implicit surfaces, Rossignac and Requicha approximate the constant radius canal surfaces by a series of smoothly connected torus sections [97]. Unfortunately, these approximations of the blends usually do not maintain a tangential contact with the faces upon which the ball is rolling, hence potentially creating gaps or self-intersections in the boundary of the result.

Furthermore, the envelope of the canal surfaces may self-intersect. These self-intersections must be detected and trimmed [60] [61].

Instead of a constant-radius blend, the designer may need to produce a variable-radius blend, where the radius of the ball varies as it rolls on A. Supporting such a functionality poses several new challenges: (1) How to precisely specify the radius variation along a single blend, (2) how to conveniently specify the radius variation for

all the blends in an entire solid or feature, and (3) how to provide an unambiguous set-theoretic definition of the resulting blended solid.

As the ball rolls on two elements of the boundary of A, the resulting blend is the subset of a variable-radius canal surface. Because the center of the ball remains equidistant from both boundary elements, it must lie on their bisecting surface. Hence, the *spine* of the canal-surface of each blend, i.e., the trajectory followed by the centers of the rolling ball, is a curve on the *Voronoi surface* (the set of points of A equidistant from two or more elements of the boundary of A, also called *medial axis surface* [84] as discussed in Chapter 2).

Pegna [87] proposed to let the designer provide an approximation of the spine and to use an automated iterative process to snap the spine to the Voronoi surfaces.

One may envision computing [61] the Voronoi surface M of A and then asking the designer to trace on it a network of spines, i.e., paths for the rounding ball. For filleting, one would use the medial surfaces of the complement $!A$ of A. Instead of tracing the spines manually, Chandru *et al.* propose to specify them as intersections of Voronoi surfaces with a reference surface C provided by the designer [22]. Chuang *et al.* [30] derive the spine from a reference curve endowed with a parametric radius function. They use a marching procedure to compute the spine and linkage curves simultaneously. Voronoi surfaces may be computed using a variety of approaches [84] [132]. To bypass the need of computing Voronoi surfaces, Chandru *et al.* propose to trace the spines numerically and to approximate the desired portions of variable-radius canal surfaces with pieces of Dupin cyclides [39] [74] [75] [89] [23], which are envelopes of one of the four families of spheres that touch three given "control" spheres. Their spines are conic sections.

Note that the *relative blending* approach proposed here is an extension of this Dupin cyclide method in two ways:

1. Two of the spheres used in the definition of a Dupin cyclide are replaced by an

arbitrary solid A.

2. The control shape B that defines the blend is not restricted to be a sphere.

The complexity of the Voronoi surfaces may appear intimidating to the designer, since they usually include many details that are not used for a specific blending. In contrast, the *relative blending* approach proposed here does not require showing the medial surfaces to the designer and uses what we believe to be a more intuitive control surface strategy. The proposed control surface B constrains the radius of the rolling ball indirectly - as the rolling-ball must remain in tangential contact with two elements of the boundary of A *and* with B. For example, pulling B away from A near a concave edge will increase the radius of the rolling-ball locally and hence increase the radius of the blend.

4.2.2 More general blends

In this section we discuss blends that are not necessarily circular. Krasauskas [66] proposed to construct Pythagorean Normal (PN)-blends between natural quadrics (cylinders, spheres, cones). These blends have two advantages over their canal-surfaces counterparts: improved shapes and lower degree surfaces. A surface is Pythagorean Normal when its normal is rational [88].

Given the implicit quadric equations $G = 0$ and $H = 0$ representing two candidate surfaces, potential methods [56] [122] define the blend as the sweep of the curve of intersection between $G = s$ and $H = t$, where s and t satisfy $f(s, t) = 0$, with four constraints: $f(a, 0) = 0$ and $f(0, b) = 0$ ensure that the first contact curve (where the blend is in tangential contact with each quadric surfaces, also called trimline or linkage curve) is the intersection of $G = a$ and $H = 0$ and the second contact curve is the intersection of $G = 0$ with $H = b$. $f'(a, 0) = (1, 0)$ and $f'(0, b) = (0, 1)$ ensure that the blend is tangent to the candidate surface along the corresponding contact curves. A conic section may be used for f. The blend is a subset of a quartic surface.

Hoffmann and Hopcroft [59] show that such blends can be constructed when the two contact curves lie on the same quadric surface Q. Typically, to construct such a blend, the designer specifies the constants a and b, which control the distance between the contact curve on one surface and the other surface. Hoffman and Hopcroft extend this approach to blending corners where three surfaces meet [57]. Alternatively the linkage curves can be computed through geodesic offsetting of the intersection curve and connected through a bicubic B-spline surface blend [8].

Filkins *et al.* [42] propose to define the contact curves first and then to either establish the spine or a correspondence between the trim lines. Then, tangents to each cross-section curves of the blend may be derived so that they are orthogonal to both the contact curve and tangent to the candidate surface upon which it lies. Finally, arcs that interpolate these boundary conditions may be constructed in various ways. For example, Partial Differential Equations (PDEs) may be used to define the blend [16]. Zhang and You [134] propose a blending method based on a fourth-order partial differential equation controlled by three vector-valued parameters. They also present a PDE method which is able to achieve C^2 continuity of the blend surface using a sixth-order PDE with one vector-valued parameter [135].

Semi-analytic formulations of a blend of a specific type of varying radius may be obtained for rounding intersections between specific pairs of surfaces. Fjällström's approach [43] replaces edges of polyhedra with cubic b-splines such that the variable radius is controlled by weights assigned to the original edges. Blends of collections of shapes represented by implicit surfaces [54] may be created using Ricci's combinations [93] of real functions [85], as well as their generalization to convolution surfaces [15] and blending functions [37]. Hierarchical representations combining R-functions in a CSG-like manner were introduced in [85] and used in [130]. Rockwood *et al.*'s [94] displacement method defines an implicit blending technique which creates a C^1 continuous blending for the entire domain of the blend. Another implicit

approach [86], based on displacement functions as well as R-functions, allows for the local selection of geometry to blend using a control shape, which is also defined as a real-valued function. This bounding control shape localizes the blend and also affects its shape but doesn't define it completely. In fact, the shape of the blend is controlled by three additional parameters: two of the parameters are weights applied to each shape and the third controls the total displacement of the blending surface. For example, compare the shape of the blend in Fig. 9-e of [86] to Fig 24. At each point of an intersection curve between two faces of A, one may consider the intersections of the two surfaces with the normal (cross-section) plane and compute a circular rounding curve in that cross-section plane. The value of the radius for the rounding may evolve as a function of arc-length or other parameterization of the intersection curve. Alternatively, a 3D sweep may be computed along the intersection curves using a parameterized 2D template to define the rounding cross-sections curves [101]. Such blends may produce undesired results (such as self-intersections of the rounding surface when, for example, filleting the intersection between a plane and a cylinder whose axis is at 60 degrees with the normal to the plane) or simply be impossible (when the two surfaces do not intersect at all) as illustrated in Fig. 24.

Shape blending can also be accomplished using iterative fairing approaches. A fairing algorithm [112] reduces the problem of blending to implementing a linear-time low-pass filter and it allows for constraining the fairing to specific portions of the input shape and supplying other constraints. The result of a fairing can be constrained to obey a maximum radius of curvature, for example.

4.3 Set-theoretic Formulation of Relative Blending

Consider two shapes, A and B. We use respectively $!A$, iA, eA, bA, and cA, to denote the complement, interior, exterior, boundary, and closure of set A. As mentioned previously, we assume that A and B are each closed-regularized. Here we redefine

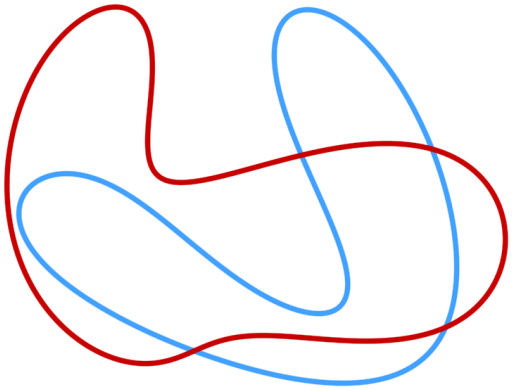
the term *moat* and symbol X from Chapter 3 for solids instead of boundaries as $(A\Delta B) \cup bA \cup bB$. Note that X is the union of the boundaries bA and bB with the symmetric difference, $A\Delta B$, of the two shapes. The boundaries are necessary since $A\Delta B$ does not include $bA \cap bB$. Let O be the set of all "maximal" balls of possibly zero radius (when A and B intersect) in X that are in tangential contact with both bA and bB. Let the *pad* P be the union of all balls of O. O and P are shown in Fig. 23 for a simple 2D example.

We define the *mean* M of A and B as the set of points (solid in 3D or planar face in 2D) closer to $iA \cap iB$ than to $!(A \cup B)$. Let C denote its boundary. Note that C is the location of the centers of balls that touch bA and bB. Except for singular situations, C is the subset of the medial axis surface of X such that all branches of the medial axis of X have been removed, and only parts of the medial axis that are equidistant from at least one point on both A and B remain.

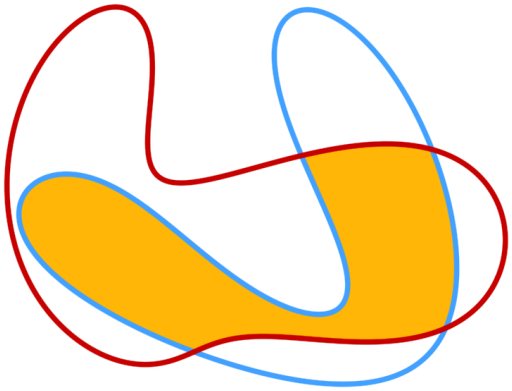
Using the pad P and mean M, we provide a set theoretic definition of the *relative blending* of A with respect to a control shape B as:

$$R_B(A) = (A \cap (M \cup P)) \cup (M - P) \tag{19}$$

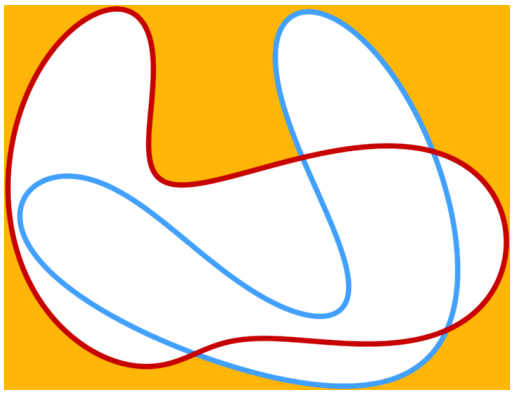
We identify the incompatible features of A and B that have been smoothed (i.e. removed) by this relative blending operation. Specifically the relative blending adds $(M - P) - A$ to A and removes $A - (M \cup P)$ from A. The following figures show the various pieces of the boolean formulation, starting with the input curves A and B and finally arriving at the relative blended solutions $R_B(A)$ and $R_A(B)$.



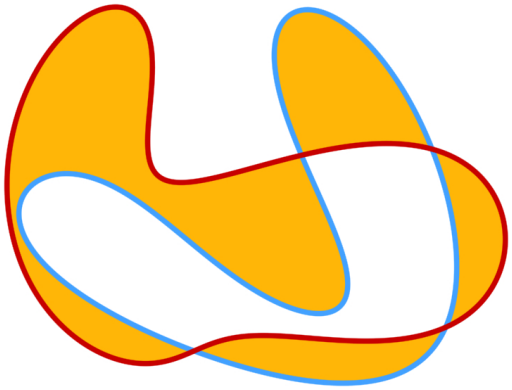
A (blue), B (red)



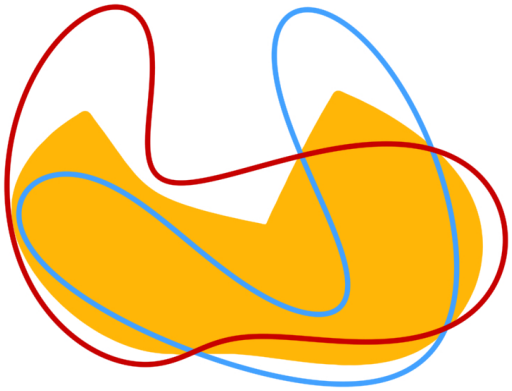
$A \cap B$



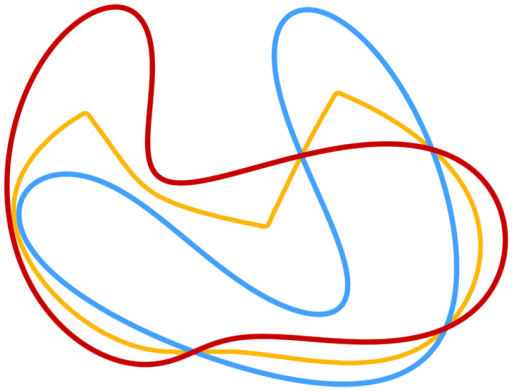
$!(A \cup B)$



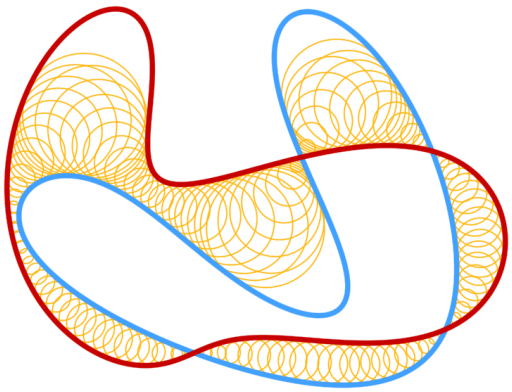
moat X



mean M

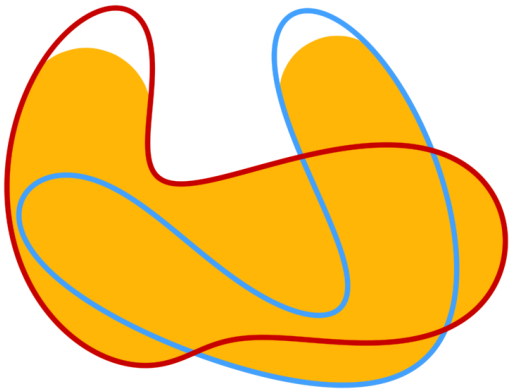


disk centers C

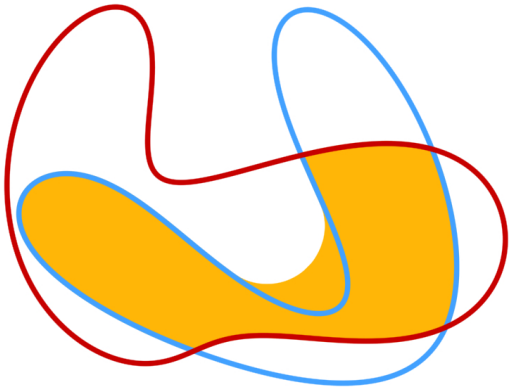


O (disks)

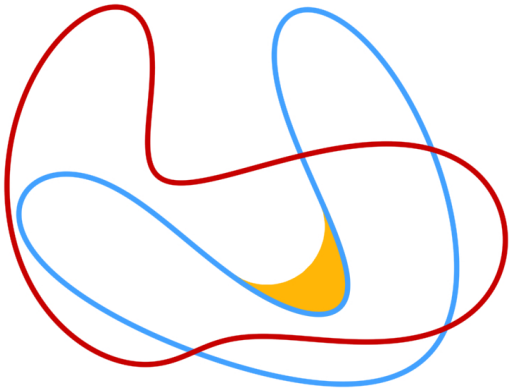
pad P



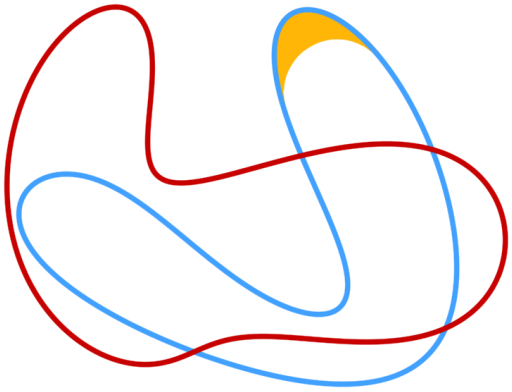
$M \cup P$



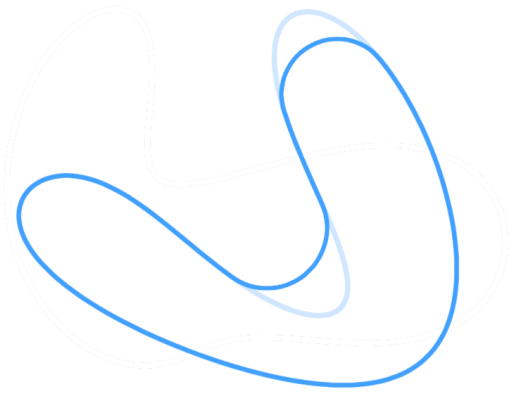
$M - P$



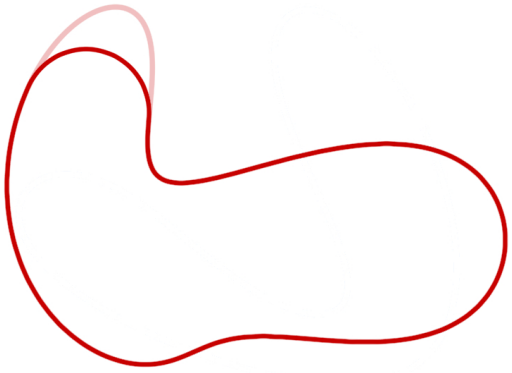
$(M - P) - A$



$A - (M \cup P)$



$R_B(A)$



$R_A(B)$

4.4 Computation

We include below an outline of two different implementations of *relative blending*. Many others are possible.

4.4.1 Medial Axis Surfaces

Because each point on the boundary C of M is equidistant from both bA and bB, C is a subset of the medial axis surface [17] of X. Several techniques have been proposed for computing the medial axis or its approximation [84].

In 2D, Montanari [82] presents the classic explicit algorithm for polygons which propagates the boundaries inward while identifying self-intersections to determine branching points on the medial axis and then connecting them with linear or parabolic segments. For sets discretized on a regular grid, Telea *et al.* [113] present an approach

which utilizes a form of fast marching methods in order to approximate a distance transform, the result of which is thresholded to produce skeleton branches.

In 3D, the medial axis is no longer a graph of 1D segments, but a complex of 2D surfaces, sometimes called the medial axis surface, or Voronoi surface. Foskey *et al.* [45] approximate the medial axis surface by analyzing distance fields computed from triangle meshes. An explicit approach for polyhedra, presented by Sherbrooke *et al.* [107] traces 1D seams from vertices and connects their intersection points with 2D sheets. More recently, Du and Qin [38] discuss extracting the medial axis using parabolic PDEs and Yang *et al.* [132] discuss a fast sampling approach to approximate points on the medial axis.

The boundary C of M may be obtained from the medial axis surface of X by trimming away portions that are not equidistant from bA and bB. Each point $\mathbf{p}$ on the boundary of M inherits the radius $r(\mathbf{p})$ defined as the distance to both bA and bB. The pad P is the union of balls of center p with radius $r(\mathbf{p})$. Once M and P are known, we compute $R_B(A)$ using Eqn. 19. (Note that the medial axis surface of X discussed above is different from the Voronoi surfaces of A and of $!A$, as used by Hoffmann in [61].)

4.4.2 Regular grids

We describe here a simple approximating approach for performing *relative blending* in 3D when A and B are CSG models involving simple primitives. This is the approach that was used to generate all of the 3D examples in this chapter.

Assume the following fields per voxel: one-bit flags a, b, p, m, x, ra to record membership in A, B, P, M, X, $R_B(A)$ and values di and de which define the distances to $A \cap B$ and $!(A \cup B)$. Our algorithm assigns these values, and hence computes a discrete approximation of the corresponding sets by performing the following simple steps illustrated in pseudo-code below.

Algorithm 2 Pseudo-code for approximating relative blending on regular grids

```
1: for all voxels s do
2:    a(s) ← A.contains(s)
3:    b(s) ← B.contains(s)
4:    x(s) ← a(s) != b(s)
5:    if x(s) then
6:       di(s) ← A.distance(s)
7:       de(s) ← B.distance(s)
8:       m(s) ← di(s) < de(s)
9:    end if
10: end for
11: for all voxels s do
12:    if x(s) then
13:       t ← voxel(s.x + 1, s.y, s.z)
14:       u ← voxel(s.x, s.y + 1, s.z)
15:       v ← voxel(s.x, s.y, s.z + 1)
16:       for all voxels w in t, u, v do
17:          if m(s) and !m(w) then
18:             p.rasterizeSphere(s, di(s))
19:             p.rasterizeSphere(w, de(w))
20:          else if !m(s) and m(w) then
21:             p.rasterizeSphere(s, de(s))
22:             p.rasterizeSphere(w, di(w))
23:          end if
24:       end for
25:    end if
26: end for
27: for all voxels s do
28:    ra(s) ←    (a(s) and (m(s) or p(s)))
                  or (m(s) and !p(s))
29: end for
```

Lines 2 - 4 first rasterize A and B and then compute the symmetric difference. Lines 6 - 8 compute the distance fields from the interior ($A \cap B$) and exterior ($!(A \cup B)$) and also set m to *true* wherever the distance to the interior is less than the distance to the exterior. Once m is computed for all voxels, the next traversal then finds every voxel $\mathbf{s}$ in x such that $m(\mathbf{s})$ is not equal to m at the three positive neighbors of $\mathbf{s}$ (one in each dimension). The purpose of this step is to find all voxels incident upon the boundary of m. At each of these incident voxels, a sphere is then rasterized into p

Table 1: Average timings for computing the *relative blend* in Fig. 24(a-c) using the binary regular grid algorithm presented in this chapter.

Dimensions	# Voxels	Running Time	Voxels/sec
32^3	$32K$	$0.0047s$	$6,972K$
64^3	$262K$	$0.038s$	$6,898K$
128^3	$2,097K$	$0.38s$	$5,518K$
256^3	$16,777K$	$4.81s$	$3,487K$
384^3	$56,623K$	$24.11s$	$2,348K$
512^3	$134,217K$	$79.85s$	$1,680K$
640^3	$262,144K$	$216.04s$	$1,213K$

with radius equal to either $di(\mathbf{s})$ or $de(\mathbf{s})$, depending on which side of the boundary the voxel lies. Finally, in lines 27 - 29, the final relative blending is computed using Equation 19.

Performance of our non-optimized implementation of the discrete relative rounding algorithm depends on the resolution and also on the nature of the model. Typically, over 50% of the cost is spent in the second loop (lines 11-26) rasterizing the spheres of the pad. Table 1 provides typical timings for the model shown in Fig. 24(a-c) at various resolutions. We list the number of voxels in each dimension, the total number of voxels, the total time, and the number of voxels processed per second.

4.5 Suggested Interface and Applications

We briefly discuss three applications of relative blending.

4.5.1 Local blend creation

In [18], Braid discusses the need for local adjusting of blends. In the application of relative blending, the designer may isolate a specific feature F of A and blend it as described above to obtain $R_B(F)$. Then, the result may be incorporated in A by computing $A\Delta(F\Delta R_B(F))$. F could be selected by letting the user specify an additional control shape D such that $A \cap D = F$. For example, in Fig. 24, D and B

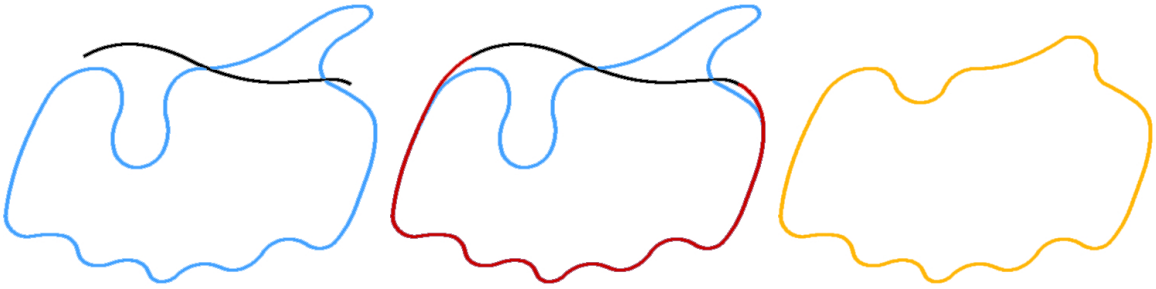


Figure 25: *Left:* Instead of specifying the entire control shape B, a curve K (black) may be used to define B locally. *Center:* The rest of B is then completed automatically by using A and "snapping" to the endpoints of K to produce a smooth shape. *Right:* This produces a final shape $R_B(A)$ that is identical to A except for the area specified by K.

are identical and shown in red.

4.5.2 Control shape construction

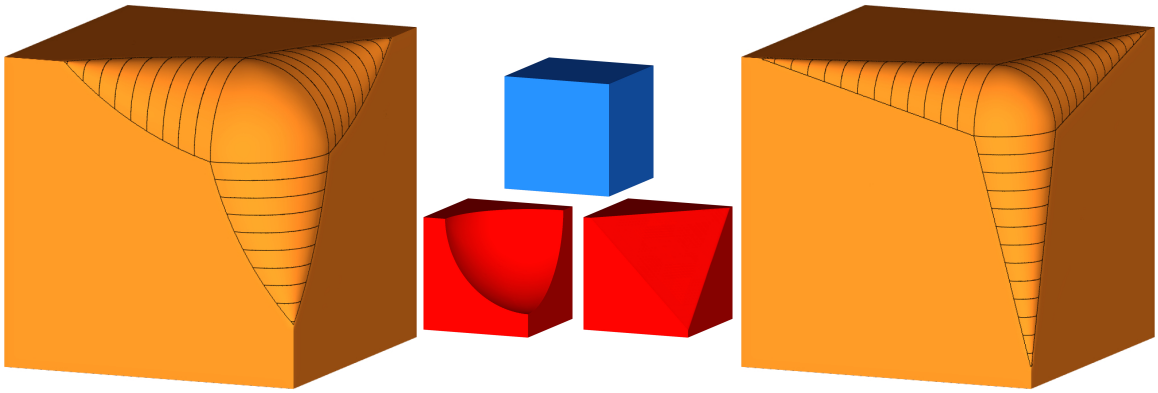


Figure 26: Two examples of a relative blending operation on a convex corner with the same input shape A (blue) and two different control shapes B (red). The two different results $R_B(A)$ are shown in orange. Note that the control shapes are actually just modified versions of A produced by a boolean subtraction with a sphere (left) and a halfspace (right).

Instead of the localization discussed above, we start by setting B to A. Clearly, relative blending in this condition will leave A unchanged (ignoring the effect of sampling if an approximating method is used). Now the designer may modify B to obtain the desired control shape. For example, in 2D, the designer may simply draw a control curve segment K as shown in Fig. 25. The portion of B that is close to K

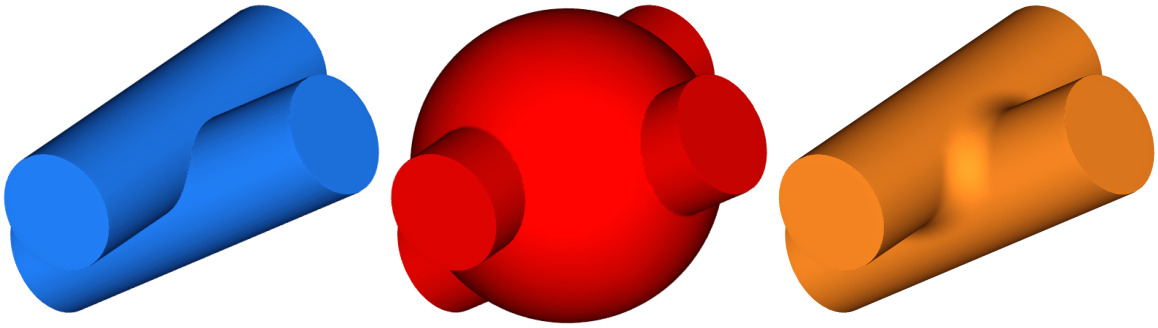


Figure 27: *Left-top:* An input shape A is the union of two intersecting cylinders. *Left-bottom:* A control shape B defined as the union of A with a sphere. *Right:* The final rounded shape $R_B(A)$.

will snap to K. We use a small range near the ends to provide a smooth transition.

In 3D, B may be precisely defined through CSG operations applied to A, as demonstrated in Figs. 27 and 28. In some application areas where the shape of the blend does not require set-theoretic semantics, we can use interactive space warps [72] of A to produce the final control shape B. This enables the designer to use a set of standard tools for specifying the control shape B and hence controlling the radius of the blend. For example, to blend only one feature locally, the user may place a sphere centered on that feature and compute the boolean difference with A to obtain B, as shown in Fig. 26.

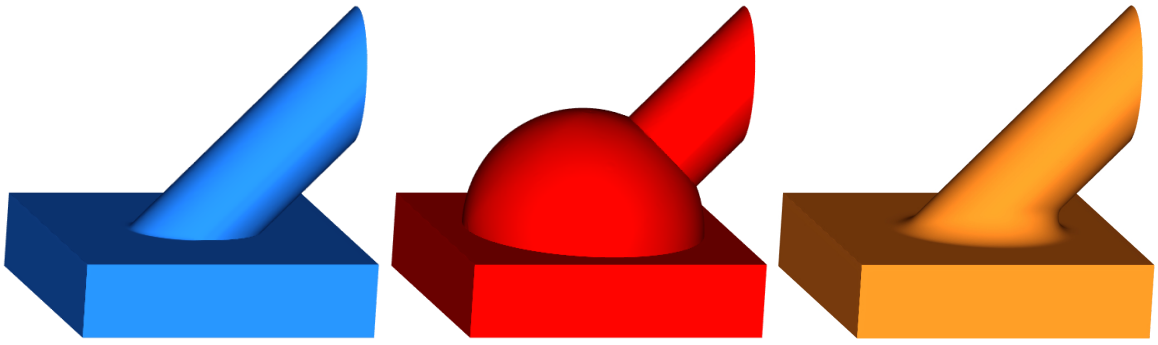


Figure 28: *Left:* An input shape A composed of a cylinder and a plane. *Center:* A control shape B defined as the union of A with a sphere. *Right:* The final rounded shape $R_B(A)$ with a variable radius fillet around the base of the cylinder. The radius of curvature is larger on the left side than on the right side.

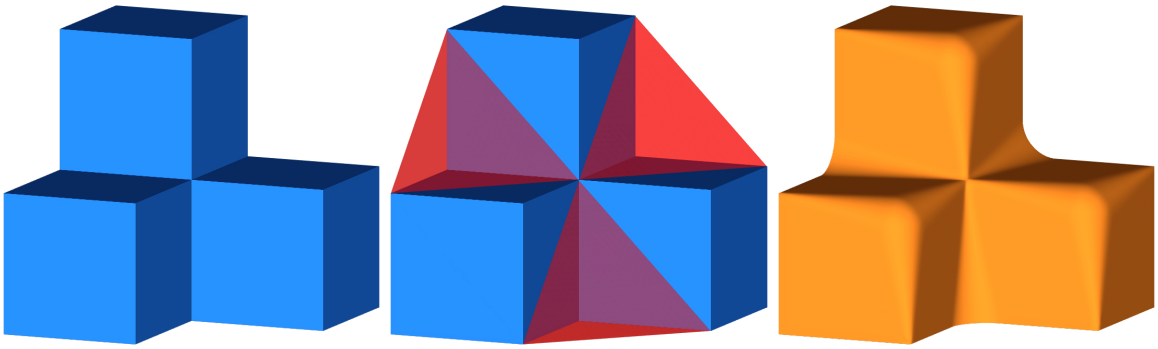


Figure 29: A shape A (blue) and a halfspace B defined by a plane (red) as the bounding shape. Note that the single *relative blending* operation results in both rounds and fillets (right).

Furthermore, note that several consecutive relative blending operations may be performed in order to round both the concave and convex corners even though the relative blending is often sufficient to round both in one operation, as shown in Fig. 29. However, this global user interface for creating blends may not be appropriate for handling the precise specs of complex blends such as those described by Braid [18].

4.5.3 B-compatibility

As discussed in Chapter 3, two shapes are *b-compatible* when each ball of O has a single contact point with A and a single contact point with B. Even though $R_A(B)$ and $R_B(A)$ are not *b-compatible*, they are *quasi-b-compatible* as the limit of a family of *b-compatible* shapes. To force A and B to be *quasi-b-compatible*, we replace them with $R_A(B)$ and $R_B(A)$. The result of this simultaneous and symmetric *relative blending* is shown in Fig. 30. Chapter 9 discusses using *relative blending* as a tool to force curves to be *quasi-b-compatible* in order to operate on non-*b-compatible* curves.

4.6 Limitations

4.6.1 Non-circular blends

Because the *relative blending* formulation included here falls into the category of ball-rolling techniques for computing blends, it is constrained to producing circular blends.

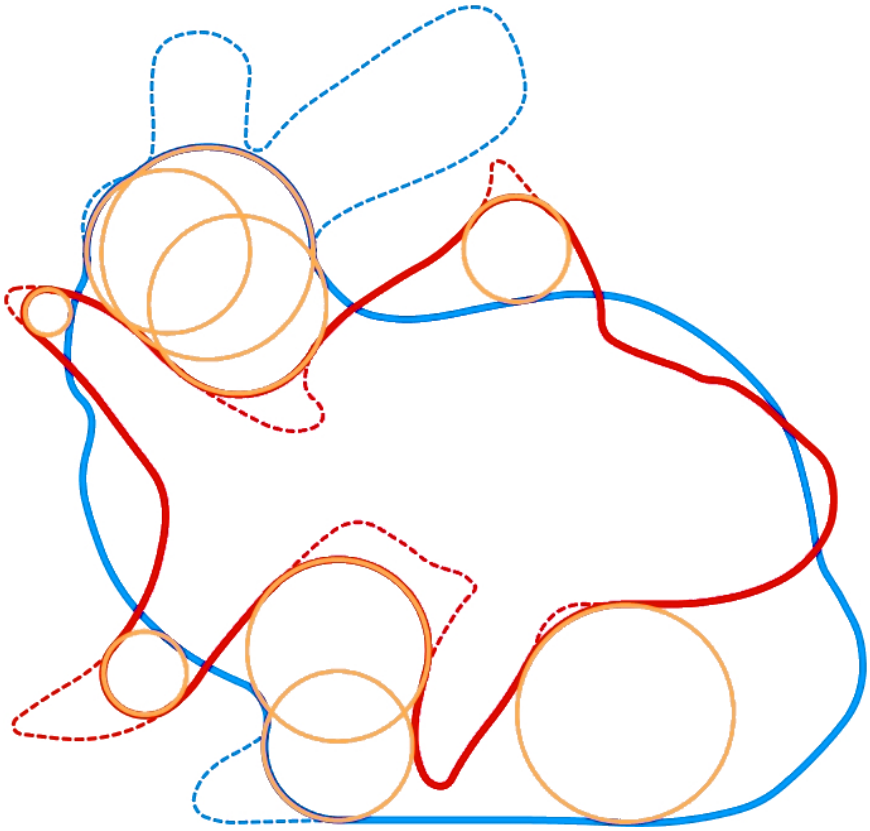


Figure 30: Figure of bunny and fish showing the disks of local maximum radii in O that create the rounding arcs for one or the other shape. The incompatible features (i.e., portions of the boundary of each shape that have been removed by our relative blending performed on both shapes) are shown in dashed line style.

If elliptical blends, for example, are desired, then there are other techniques, such as those described in Section 4.2.2, to better accomplish this task.

4.6.2 Precise edits

Our suggested interface for *relative blending* may not be appropriate for complex blends that require extreme precision. There are no parameters to *relative blending* other than the geometry of the control shape. For this reason, making precise edits to the result of a blend by tweaking the control shape may be tedious within this given framework.

4.6.3 Smoothness

The result of *relative blending*, although smooth in the general case, may produce sharp edges just as in other rolling-ball techniques. This can happen when different sweeps of balls intersect, thus creating sharp geometry where the two sweeps "pinch" together.

CHAPTER V

TANGENT-BALL CORRESPONDENCE

This chapter discusses the correspondence derived from *tangent-ball* contact and also the algorithms for computing tangent balls. These algorithms are used to discretely sample the interior medial axis of the *moat* X of two curves A and B. They are also used to determine *b-compatibility*, as discussed in Chapter 3.

5.1 Ball-Map

Consider the maximal ball centered at point $\mathbf{m} \in M$. The maximal disk O centered at $\mathbf{m}$ touches A at $\mathbf{a}$ and B at $\mathbf{b}$, as shown in Figure 31. The *ball-map* [24] establishes the correspondence between the closest projection $\mathbf{a}$ of $\mathbf{m}$ onto A and the closest projection $\mathbf{b}$ of $\mathbf{m}$ onto B.

The *ball-map* may be viewed as a continuous version of an approach proposed by [64] for establishing correspondences between curves by considering their distance fields. They use discrete distance fields computed on a regular grid for the two curves A and B. The points $\mathbf{m}$ of M are then defined where adjacent pixels are closest to

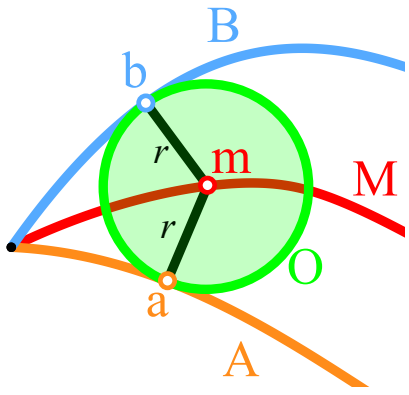


Figure 31: A *ball-map* between points $\mathbf{a}, \mathbf{b}$ on curves A, B centered at point $\mathbf{m}$ on the internal medial axis M of the *moat* X.

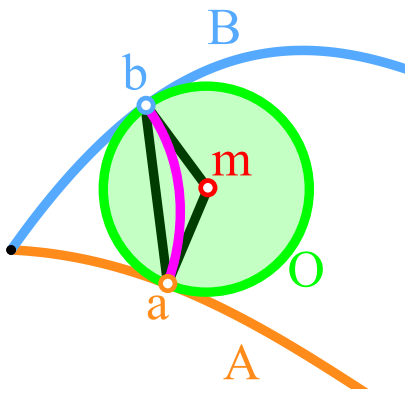


Figure 32: A *ball-map* circular arc (violet) inscribed in the triangle $\triangle$**amb**.

different curves.

A uniform sampling of the *ball-map* correspondence may be computed in several ways: (1) By initially computing M as the medial axis of the *moat* X between two curves A and B using efficient medial axis construction techniques [45][132] and then generating the closest projections **a** and **b** for a set of uniformly spaced sample points $\mathbf{m} \in M$; (2) By computing the radii of the maximal disks that touch A at a set of uniformly spaced samples **a**; or (3) By simultaneously advancing the corresponding points, **a** and **b**, until one of them has travelled from the previous sample on its curve by a prescribed geodesic distance.

5.1.1 Ball-map Arc

Given a *ball-map* centered at **m** and tangent to **a** and **b**, the points $\mathbf{a}, \mathbf{m}$ and **b** form an isosceles triangle as shown in Figure 32. Within such a triangle, we can inscribe a circular arc that is orthogonal to A at **a** and B at **b**. We associate this circular arc to the *ball-map* correspondence pair.

5.2 Details of the Ball-map construction for PCCs

We include here the details of an exact implementation (except for numerical round-off errors) for the case of piecewise-circular curves in 2D, where A and B are each a series of smoothly connected circular-arc edges.

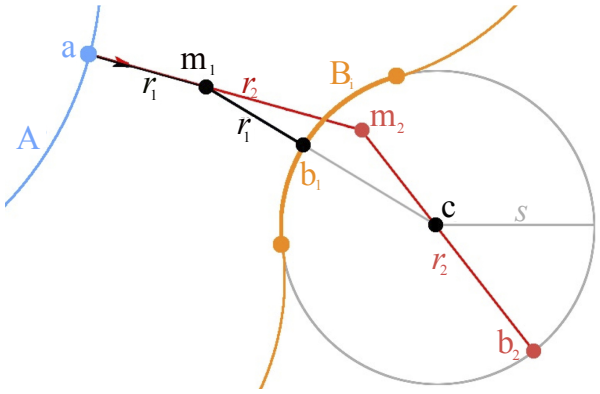


Figure 33: Computing r, $\mathbf{m}$ and $\mathbf{a}$ from $\mathbf{b}$ for a circular arc B_i. The formula produces two candidate points $\mathbf{b}_1, \mathbf{b}_2$. In this example, $\mathbf{b}_2$ is discarded since it does not lie on the arc B_i.

We compute the corresponding point $\mathbf{b}$ from $\mathbf{a}$, assuming $\mathbf{a}$ is not a point of intersection of A and B, as follows. Consider the parameterized offset point $\mathbf{m} = r\widehat{\mathbf{N}}_A(\mathbf{a})$, whose distance from $\mathbf{a}$ is defined by the parameter r. Here, we have oriented $\widehat{\mathbf{N}}_A(\mathbf{a})$ so that it points towards the interior of the *moat* X. $\mathbf{m}$ is the center of a circle of radius r that is tangent to A at $\mathbf{a}$. We want to compute the smallest positive r for which $\mathbf{m}$ is at distance r from B, and hence for which the circle is tangent to B.

First consider a circular edge B_i of B with center $\mathbf{c}$ and radius s (Fig. 33). We compute r_1 and r_2 as the roots

$$\frac{s^2 - \overrightarrow{\mathbf{ca}}^2}{2\widehat{\mathbf{N}}_A(\mathbf{a}) \cdot \overrightarrow{\mathbf{ca}} \pm 2s} \tag{20}$$

of

$$\overrightarrow{\mathbf{cm}}^2 = (r \pm s)^2 \tag{21}$$

and keep the smallest positive solution for which the *ball-map* tangent point ($\mathbf{b}_1$ or $\mathbf{b}_2$) lies on B_i.

We apply the above approach to all edges B_i of B. We compute the r-value for a circle supporting each edge, compute the corresponding candidate point $\mathbf{b}$ on the circle, discard it if it is not contained within the arc (such as $\mathbf{b}_2$ in Figure 33), and select amongst the retained $(r, \mathbf{b})$ pairs with the smallest r-value.

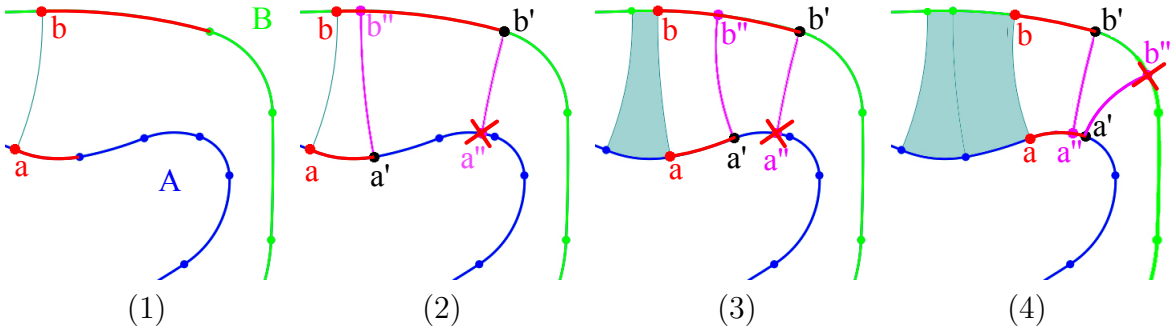


Figure 34: The lacing algorithm for *b-compatible* PCCs. (1) An initial mapping is given between 2 points $\mathbf{a}, \mathbf{b}$. They are vertices and sit at the start of a circular arc edge segment (red). (2) A *ball-map* is computed from the endpoints of the current arcs $(\mathbf{a}', \mathbf{b}')$ to the current arc on the other curve (red). Only $\mathbf{a}'$ produces a valid *ball-map* to the edge segment $\mathbf{b}, \mathbf{b}'$, so only it is kept. (3) $\mathbf{a}$ and $\mathbf{b}$ step forward to the positions of the previous *ball-map* and the process repeats. Again, only $\mathbf{a}'$ produces a valid *ball-map*, so only it is kept. (4) $\mathbf{a}$ and $\mathbf{b}$ again step forward. This time, $\mathbf{b}'$ produces the valid *ball-map*.

Since we assume that A and B are *b-compatible*, there is exactly one $(r, \mathbf{b})$ pair for each point $\mathbf{a} \in A$. The above process computes the *ball-map* correspondence for any desired sampling of A or B.

5.2.1 Lacing acceleration for PCCs

To accelerate the computation of the *ball-map* for *b-compatible* PCCs and produce a sampling-independent representation from which different sampling densities can be quickly derived, we perform a "lacing" process (Fig. 34), to split the *moat* into *arc-quads*, each bounded by 4 circular arcs: one being a circular-arc edge-segment of A, one being a circular-arc edge-segment of B, and two being *ball-map* arcs from a vertex of A or B to its image on the other curve.

To perform the lacing, we first pick a vertex $\mathbf{a} \in A$, where two edges of A meet and compute its image $\mathbf{b} \in B$ as described above. Then, we perform a synchronized walk to "lace" the *moat*, one vertex of A or B at a time. At each step, $\mathbf{a}$ is the start of an edge-segment A_i of A not yet laced and $\mathbf{b}$ is the start of an edge-segment B_k of B not yet laced. Let $\mathbf{a}'$ be the end of A_i and $\mathbf{b}'$ be the end of B_k. Let $\mathbf{b}''$ be the

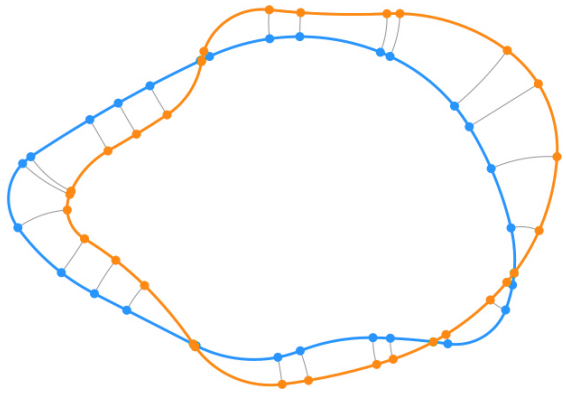


Figure 35: Lacing splits the *moat* X into *arc-quads*. Each one is bounded by 4 circular arcs (2 edge-segments and 2 *ball-map* arcs).

corresponding point for $\mathbf{a}'$ and $\mathbf{a}''$ be the corresponding point for $\mathbf{b}'$. If $\mathbf{a}''$ falls on A_i, we record that the edge-segment $[\mathbf{b}, \mathbf{b}']$ of B_k maps to the edge-segment $[\mathbf{a}, \mathbf{a}'']$ of A_i, close the current *arc-quad* with the arc from $\mathbf{b}'$ to $\mathbf{a}''$, and set $\mathbf{a}$ to $\mathbf{a}''$ and $\mathbf{b}$ to $\mathbf{b}'$ to continue the lacing process, as shown in Fig. 34.

This lacing process splits the edges of A and B into edge-segments and establishes a bijective mapping between edge-segments of A and edge-segments of B that bound the same *arc-quad*. The cost of this pre-computation is $O(n)$ in the number of edges in A and B since at every step, only the current arc-edges A_i and B_i are used to compute *ball-map* correspondences. It can be performed in real-time, as the curves are edited, which is convenient for the interactive design of *b-compatible* curves.

The pseudo-code for the linear-time algorithm is shown below in Algorithm 3.

5.2.2 Fast b-compatibility test for PCCs

The lacing algorithm is also used in order to determine if two curves are *b-compatible* in linear time, improving on the $O(n^4)$ algorithm in Chapter 3. In order to do this, the lacing process is computed as above, however, if at any step there does not exist a *ball-map* solution from point $\mathbf{a}$ or $\mathbf{b}$ to B_i or A_i, respectively, then the curves are not *b-compatible*. The updated algorithm is shown below in Algorithm 4 where lines 29-30 simply return $false$ if a valid mapping is not found.

Algorithm 3 Pseudo-code for computing the *ball-map* between two piecewise-circular curves A and B using our fast $O(n)$ algorithm. It assumes that one set of closest points a and b are given (if A and B intersect, $a = b$). It also assumes that A and B are oriented in the same direction.

```
1: ballmaps ← ∅
2: ballmaps.add(a, b)
3: {Split each arc at the given points a and b, making a and b vertices}
4: splitArc(a.arc, a)
5: splitArc(b.arc, b)
6: {If a is a vertex then a.arc is the arc following a}
7: startA ← a.arc
8: startB ← b.arc
9: repeat
10:    {Compute a ball tangent to a.arc at a.next (next vertex of A) and b.arc}
11:    {Returns null if no solution or infinite solutions}
12:    Ba = tangentBall(a.next, a.arc, b.arc)
13:
14:    Bb = tangentBall(b.next, b.arc, a.arc)
15:
16:    {Move a and b forward depending on which maximal ball is not null}
17:    if Ba ≠ null and Bb = null then
18:       a ← a.next
19:       b ← tangentPoint(b.arc, Bb)
20:       {Split b.arc at b, such that b is now a vertex}
21:       splitArc(b.arc, b)
22:       ballmaps.add(a, b)
23:    else if Ba = null and Bb ≠ null then
24:       a ← tangentPoint(a.arc, Ba)
25:       {Split a.arc at a, such that a is now a vertex}
26:       splitArc(a.arc, a)
27:       b ← b.next
28:       ballmaps.add(a, b)
29:    else if Ba ≠ null and Bb ≠ null then
30:       {This means that Ba = Bb}
31:       a ← a.next
32:       b ← b.next
33:       ballmaps.add(a, b)
34:    end if
35: until (startA = a.arc and startB = b.arc) or END_OF_CURVE
36: return ballmaps
```

Algorithm 4 Pseudo-code for testing *b-compatibility* between two piecewise-circular curves A and B using our faster O(n) algorithm. It assumes that one set of closest points **a** and **b** are given (if A and B intersect, **a** = **b**). It also assumes that A and B are oriented in the same direction.

```
1: {Split each arc at the given points a and b, making a and b vertices}
2: splitArc(a.arc, a)
3: splitArc(b.arc, b)
4: {If a is a vertex then a.arc is the arc following a}
5: startA ← a.arc
6: startB ← b.arc
7: repeat
8:    {Compute a ball tangent to a.arc at a.next (next vertex of A) and b.arc}
9:    {Returns null if no solution or infinite solutions}
10:   Ba = tangentBall(a.next, a.arc, b.arc)
11:
12:   Bb = tangentBall(b.next, b.arc, a.arc)
13:
14:   {Move a and b forward depending on which maximal ball is valid}
15:   if Ba ≠ null and Bb = null then
16:      a ← a.next
17:      b ← tangentPoint(b.arc, Bb)
18:      {Split b.arc at b, such that b is now a vertex}
19:      splitArc(b.arc, b)
20:   else if Ba = null and Bb ≠ null then
21:      a ← tangentPoint(a.arc, Ba)
22:      {Split a.arc at a, such that a is now a vertex}
23:      splitArc(a.arc, a)
24:      b ← b.next
25:   else if Ba ≠ null and Bb ≠ null then
26:      {This means that Ba = Bb}
27:      a ← a.next
28:      b ← b.next
29:   else
30:      return false
31:   end if
32: until (startA = a.arc and startB = b.arc) or END_OF_CURVE
33: return true
```

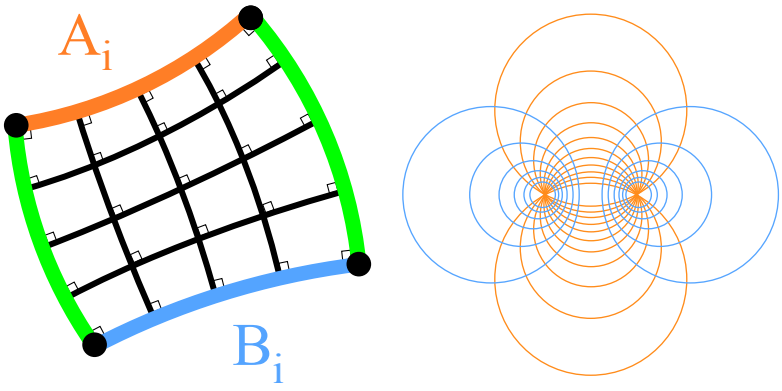


Figure 36: Left: Lacing splits the *moat* X into *arc-quads*. Each one is bounded by 4 circular arcs (2 corresponding edge-segments A_i, B_i and 2 *ball-map* arcs (green)) and defines a set of orthogonal circular arcs in the interior (black). Right: The orthogonal circular arcs correspond to the Apollonian families of circles.

5.2.3 Arc-quads and Apollonian Circles

The arc-quads produced by lacing of PCCs, which are bounded by 4 circular arcs that are orthogonal to each other at their intersections, define two infinite sets of orthogonal circular arcs within their bounds. Figure 36-left shows an example set of these orthogonal arcs. These arcs correspond to a special family of circles discovered by Apollonius of Perga (same as from Chapter 2) known as Apollonian circles. Apollonian circles are two familes of circles such that every circle in one family intersects every circle in the other family orthogonally, as shown in Figure 36-right. In Figure 36-left, the arcs between the two green *ball-map* arcs define one family, and the arcs between A_i and B_i define the other family.

5.3 Details of Ball-map construction for PLCs

The piecewise-linear curve is a popular curve representation due to its ability to approximately represent a variety of types of curves (i.e. polygons, subdivision curves). As discussed in Chapter 3, piecewise-linear curves cannot be *b-compatible*. However, we propose here an approach for computing an approximate *ball-map*, treating piecewise-linear curves as approximations of smooth curves.

There are several possibilities for handling PLCs:

1. Use relative blended (Chapter 4) versions of each curve as approximations
2. Use PLCs as control polygons for constructing PCCs according to [99]
3. Treat vertices as circular arcs with infinitely small radii and ignore incompatibilities as long as adjacent *ball-maps* do not intersect

The first two options are then handled using the PCC method described above. Option 3 allows for working directly with the input curves instead of adding an additional level of approximation. We discuss option 3 here, including implementation details and issues of sampling.

5.3.1 Equations

First we show the formulation for computing *ball-maps* between vertex-vertex pairs and vertex-edge pairs. We compute the corresponding point $\mathbf{b}$ from $\mathbf{a}$ as follows, assuming $\mathbf{a}$ is not a point of intersection between A and B. We have the normal $\widehat{\mathbf{N}}_A(\mathbf{a})$, computed as the weighted average of the normals of the two incident edges, such that it points towards the interior of the *moat*. First, for every vertex $\mathbf{b} \neq \mathbf{a}$ of B, we compute the r-value for a ball with center $\mathbf{m} = \mathbf{a} + r\widehat{\mathbf{N}}_A(\mathbf{a})$ such that $|\mathbf{m} - \mathbf{a}| = |\mathbf{m} - \mathbf{b}| = r$ as follows:

$$r = -\frac{\overrightarrow{\mathbf{ba}}^2}{2\overrightarrow{\mathbf{ba}} \cdot \widehat{\mathbf{N}}_B(\mathbf{b})} \tag{22}$$

Notice that we don't check that $\mathbf{m}$ lies along any particular normal of $\mathbf{b}$ or even that it is within the fan of normals defined by interpolating the normals of the incident edges. Since we next compute the *ball-map* for those incident edges, one of those maps will always return a smaller radius r if $\mathbf{m}$ does not lie in the desired fan of normals.

For every edge B_i of B with oriented edge-normal $\widehat{\mathbf{N}}_B(B_i)$ and vertices $\mathbf{c}, \mathbf{d}$, we compute the r-value for a ball with center $\mathbf{m} = \mathbf{a} + r\widehat{\mathbf{N}}_A(\mathbf{a}) = \mathbf{b} + r\widehat{\mathbf{N}}_B(B_i)$ as follows:

$$r = \frac{\overrightarrow{\mathbf{ca}} \cdot \widehat{\mathbf{N}}_B(B_i)}{1 - \widehat{\mathbf{N}}_A(\mathbf{a}) \cdot \widehat{\mathbf{N}}_B(B_i)} \tag{23}$$

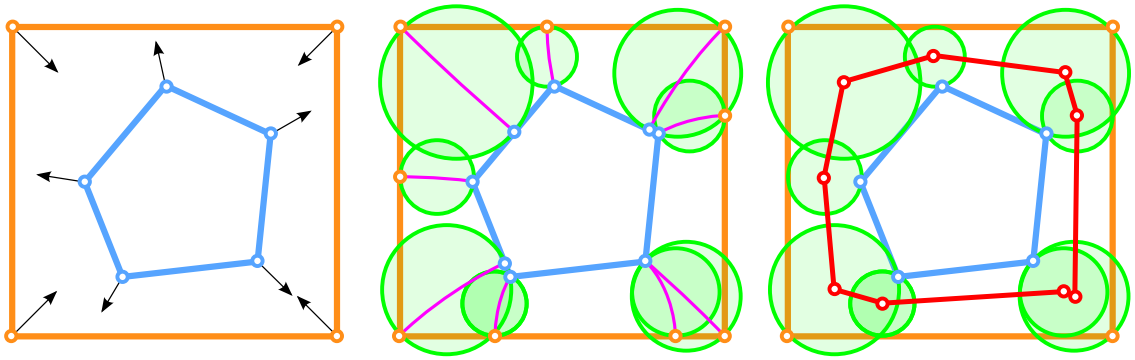


Figure 37: Construction of a non-intersecting *ball-map* for simple polygons.

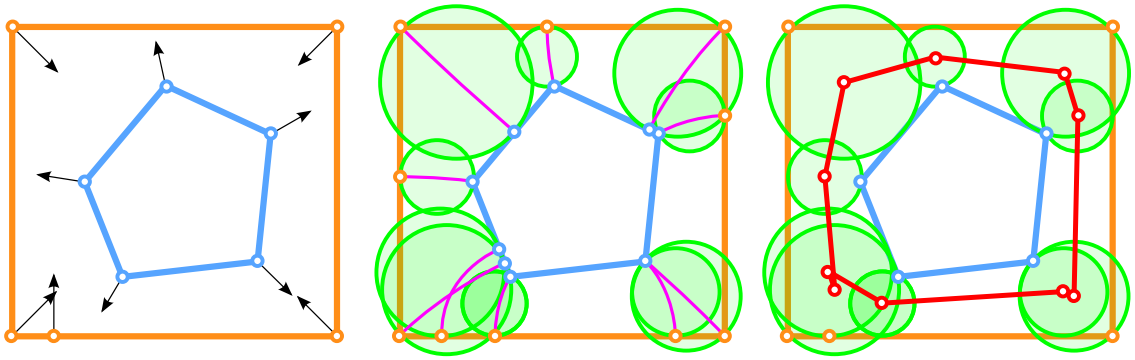


Figure 38: Construction of a self-intersecting *ball-map* for simple polygons.

The corresponding point $\mathbf{b}$ is then computed as:

$$\mathbf{b} = \mathbf{a} + r\widehat{\mathbf{N}}_A(\mathbf{a}) - r\widehat{\mathbf{N}}_B(B_i) \tag{24}$$

The candidate mapping is discarded if $\mathbf{b}$ lies outside the bounds of B_i. The minimum r ball-map candidate among all vertex-vertex and retained vertex-edge mappings is then selected for point $\mathbf{a}$. Note that care needs to be taken to ensure that the denominator of these equations is not 0 which occurs when $\mathbf{a} = \mathbf{b}$ in equation 22 or when $\widehat{\mathbf{N}}_A(\mathbf{a})$ and $\widehat{\mathbf{N}}_B(\mathbf{b})$ are parallel and oriented in the same direction in equation 23.

5.3.2 Sparsely sampled PLCs

Figure 37 shows a simple *ball-map* construction for the case of a square and a pentagon. On the left, the vertex normals are shown for every vertex. These normals are the length-weighted averages of the incident edge normals. Fig. 37 (center) shows that

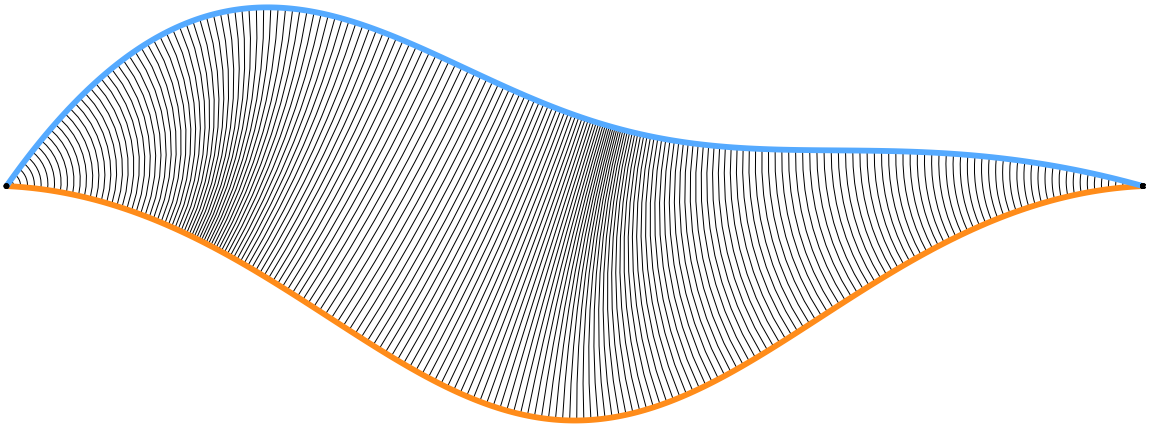


Figure 39: A sample *ball-map* result of the PLC lacing algorithm

none of the *ball-map* circular-arc trajectories computed from each vertex intersect. However, notice that some of the balls are not entirely contained within the *moat* X, which means that their centers are not actually samples on the medial axis of X. Also notice that the two *ball-maps* in the bottom-right corner share a vertex on the blue polygon, meaning that the *ball-map* construction is not symmetric at this point. In Fig. 37 (right), the median curve M is constructed by connecting the ball centers with linear edges. It has no self-intersections, despite the highlighted problems, since all the *ball-maps* are disjoint.

Now, if we add one additional vertex to the same example, as shown in Figure 38, the result becomes undesirable. Fig. 38 (center) shows that two adjacent *ball-map* circular-arc trajectories intersect, causing the median curve M to self-intersect (Fig. 38 (right)). This is to highlight that computing the *ball-map* for PLCs is very sensitive to sampling and care must be taken to avoid/resolve these issues when dealing with sparsely-sampled PLCs.

5.3.3 Smooth PLCs

The *ball-map* construction for PLCs is better behaved when the input PLCs are smooth and densely sampled. In fact, all results in this thesis were computed using the PLC method described here unless explicitly stated otherwise. Figure 39 shows

a *ball-map* result using densely sampled subdivision curves as the input. A modified form of the PCC lacing algorithm is used to compute this result, as detailed below.

Recall that in the PCC lacing algorithm, two points ($\mathbf{a}$ and $\mathbf{b}$) walk on each curve, where the maximum step-size is determined by the length of the current circular-arc. Instead, we now set a constant step-size *dStep*. For the examples in this thesis, we use a value of *dStep* that is double the maximum edge-length of both curves. In practice, this value produces desirable results. If *dStep* is not sufficiently larger than the edge-lengths, then the problems highlighted in Figure 38 are more likely to appear.

To perform the PLC lacing, we first pick a vertex $\mathbf{a} \in A$, where two edges of A meet and compute its image $\mathbf{b} \in B$ as described above. Then, we perform a synchronized walk to "lace" the gap as with the PCCs. At each step, $\mathbf{a}$ and $\mathbf{b}$ are the current positions on each curve, and could be vertices or points on edges. Let $\mathbf{a}'$ be the point obtained by walking on A a distance of *dStep* and $\mathbf{b}'$ be the sample obtained by walking on B a distance of *dStep*. Let A_i be a subcurve of A starting at $\mathbf{a}$ and having length $dStep * 1.1$. Let B_i be a subcurve of B starting at $\mathbf{b}$ and having length $dStep * 1.1$. Let $\mathbf{b}''$ be the corresponding point for $\mathbf{a}'$ and $\mathbf{a}''$ be the corresponding point for $\mathbf{b}'$. If $\mathbf{a}''$ falls on A_i, we record that the subcurve $[\mathbf{b}, \mathbf{b}']$ of B_k maps to the subcurve $[\mathbf{a}, \mathbf{a}'']$ of A_i, and set $\mathbf{a}$ to $\mathbf{a}''$ and $\mathbf{b}$ to $\mathbf{b}'$ to continue the lacing process.

The justification for A_i and B_i having length $dStep * 1.1$ as opposed to *dStep* is to add an error tolerance for dealing with numerical inaccuracies. At some steps, both $\mathbf{a}$ and $\mathbf{b}$ may yield *ball-map* solutions. If this occurs (line 28), then the result that is geodesically closest to the current values of $\mathbf{a}$ and $\mathbf{b}$ is used.

Pseudo-code for the PLC lacing algorithm is shown below in Algorithm 5. As shown in line 34, approximate *b-compatibility* is tested simultaneously.

5.4 Details of Ball-map construction for triangle meshes in 3D

As with PLCs, the ball-map construction can be approximated for triangle meshes. However, this construction also suffers from sampling issues. For every point $\mathbf{a}$ on mesh A, there are three ball-map computations that must be performed. First, assuming $\mathbf{a} \notin B$ and we have the surface normal $\widehat{\mathbf{N}}_A(\mathbf{a})$ such that it points towards the interior of the *moat* X, we compute a candidate point $\mathbf{b}$ and r-value for each vertex of B using Equation 22.

Next, we compute a candidate r and point $\mathbf{b}$ for every edge B_i. Because edges in 3D do not have a single normal, but rather an infinite fan of normals, the candidate r value must be computed as the root(s) of a quadratic equation. For every edge B_i with endpoints $\mathbf{c}, \mathbf{d}$ and normalized tangent $\widehat{\mathbf{cd}}$, we compute r as the root(s) of:

$$((\widehat{\mathbf{N}}_A(\mathbf{a}) \times \widehat{\mathbf{cd}})^2 - 1)r^2 + 2((\widehat{\mathbf{N}}_A(\mathbf{a}) \times \widehat{\mathbf{cd}}) \cdot (\overrightarrow{\mathbf{ca}} \times \widehat{\mathbf{cd}}))r + (\overrightarrow{\mathbf{ca}} \times \widehat{\mathbf{cd}})^2 = 0 \qquad (25)$$

The point(s) $\mathbf{m} = \mathbf{a} + r\widehat{\mathbf{N}}_A(\mathbf{a})$ are then projected onto the line through B_i to compute $\mathbf{b}$. If $\mathbf{b}$ does not lie on edge B_i, the candidate map is discarded. Notice as with the vertex case in 2D, we need not check that $\mathbf{m}$ lies along the appropriate fan of normals.

Finally, we compute a candidate r and $\mathbf{b}$ for every triangle $\triangle B_i$ with vertices $\mathbf{c}, \mathbf{d}, \mathbf{e}$ and normal $\widehat{\mathbf{N}}_B(\triangle B_i)$ with the following equation:

$$r = \frac{\overrightarrow{\mathbf{ca}} \cdot \widehat{\mathbf{N}}_B(\triangle B_i)}{1 - \widehat{\mathbf{N}}_A(\mathbf{a}) \cdot \widehat{\mathbf{N}}_B(\triangle B_i)}, \qquad (26)$$

which is equivalent to equation 23. The corresponding point $\mathbf{b}$ is then be computed as:

$$\mathbf{b} = \mathbf{a} + r\widehat{\mathbf{N}}_A(\mathbf{a}) - r\widehat{\mathbf{N}}_B(\triangle B_i) \qquad (27)$$

Points $\mathbf{b}$ which lie outside of triangle $\triangle B_i$ are discarded. The minimum r ball-map candidate among all vertex-vertex, vertex-edge and vertex-triangle mappings is then selected for point $\mathbf{a}$.

Algorithm 5 Pseudo-code for computing the *ball-map* between two piecewise-linear curves A and B using our fast O(n) algorithm. It assumes that one set of closest points **a** and **b** are given (if A and B intersect, **a** = **b**). It also assumes that A and B are oriented in the same direction and that A and B are smooth and densely sampled.

1: $ballmaps \leftarrow \emptyset$
2: $ballmaps.add(a, b)$
3: {**If** a **is a vertex then** $a.edge$ **is the edge following** a}
4: $startA \leftarrow a.edge$
5: $startB \leftarrow b.edge$
6: **repeat**
7: {**Get a subcurve starting at** a **with length** $dStep * 1.1$}
8: $subcurveA \leftarrow A.subCurve(a, a + dStep * 1.1)$
9: $subcurveB \leftarrow B.subCurve(b, b + dStep * 1.1)$
10: {**Get the next point by stepping forward by** $dStep$ **from** a}
11: $aNext \leftarrow A.step(a, dStep)$
12: $bNext \leftarrow B.step(b, dStep)$
13: $aNorm \leftarrow A.normal(aNext)$
14: $bNorm \leftarrow B.normal(bNext)$
15: {**Compute smallest *ball-map* from** $aNext$ **to all vertices/edges on** $subcurveB$}
16: {**Returns** *null* **if no solution**}
17: $Ba = ballMap(aNext, aNorm, subcurveB)$
18: $Bb = ballMap(bNext, bNorm, subcurveA)$
19: {**Move** a **and** b **forward depending on which *ball-map* is valid**}
20: **if** $Ba \neq null$ **and** $Bb = null$ **then**
21: $a \leftarrow aNext$
22: $b \leftarrow Ba.destination$
23: $ballmaps.add(a, b)$
24: **else if** $Ba = null$ **and** $Bb \neq null$ **then**
25: $a \leftarrow Bb.destination$
26: $b \leftarrow bNext$
27: $ballmaps.add(a, b)$
28: **else if** $Ba \neq null$ **and** $Bb \neq null$ **then**
29: {**Return the closest result**}
30: $a \leftarrow nearest(Bb.destination, aNext)$
31: $b \leftarrow nearest(Ba.destination, bNext)$
32: $ballmaps.add(a, b)$
33: **else**
34: **return** $NON\_COMPATIBLE$
35: **end if**
36: **until** ($startA = a.edge$ **and** $startB = b.edge$) **or** END_OF_CURVE
37: **return** $ballmaps$

CHAPTER VI

SHAPE DISCREPANCY MEASURES BASED ON TANGENT BALLS

This chapter defines several possible discrepancy measures based on tangent balls and compares them to the classic measures of Hausdorff distance and Fréchet distance.

Recall that the Hausdorff distance (Chapter 2) between two shapes A and B is computed as $H(A,B) = \max(\max_{\mathbf{a}\in A}(d(\mathbf{a},B)), \max_{\mathbf{b}\in B}(d(\mathbf{b},A)))$, and utilizes only point-to-shape distance computations. Fréchet distance [4] is another discrepancy measure between two curves which uses a continuous mapping from one curve to the other. Given two curves A and B with parameters $s,t \in [0,1]$, respectively, and a mapping $m : [0,1] \rightarrow [0,1]$, the parametric distance is computed as $D_p(A,B,m) = \max_{t\in[0,1]} ||A(t) - B(m(t))||$. The Fréchet distance is defined as the minimum D_p for all possible mappings m: $F(A,B) = \min_{\forall m} D_p(A,B,m)$.

Fig. 40 shows an example where the Hausdorff distance does not reflect how different the curves actually are. The Fréchet distance better captures this discrepancy.

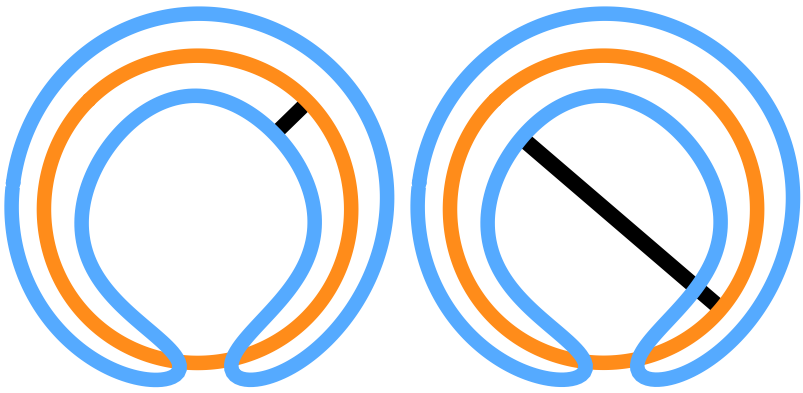


Figure 40: Sometimes the Hausdorff distance can fail to capture the magnitude of the discrepancy (left). The Fréchet distance (right) does a better job of highlighting the discrepancy between these two curves.

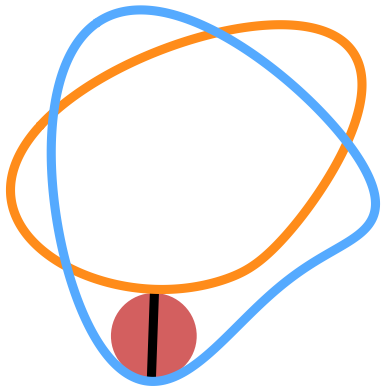


Figure 41: When curves are *b-compatible*, the Hausdorff, Fréchet, and *ball-distance* are identical, as depicted by the black line.

6.1 Ball Measures

6.1.1 Ball Distance

We propose a maximum discrepancy measure between two shapes called the *ball-distance*. The *ball-distance* between two shapes can be measured using the L_∞ norm on the diameter of the maximal balls in the *moat* X. In particular, it has been shown [24] that when the input curves are *b-compatible*, using the L_∞ norm computes both the Hausdorff [7] and the Fréchet distances between these two sets, as shown in Fig. 41 where all three are identical.

If the input curves are not *b-compatible*, then the *ball-distance* corresponds to the L_∞ norm on the diameter of the maximal balls in the *moat* of $R_B(A)$ and $R_A(B)$ (the relative blended versions of A and B as defined in Chapter 4).

The L_1 or L_2 norms on the diameter of the maximal balls or the travel distance may also be used to identify regions of discrepancy. The travel distance measure accentuates orientation differences. Both may be used locally to identify and visualize areas with large discrepancy between curves or surfaces, or globally.

6.1.2 Ball Discrepancy

One may declare that a shape A is not an acceptable realization of a nominal shape B if either A or B have incompatible features. If incompatible features exist, the

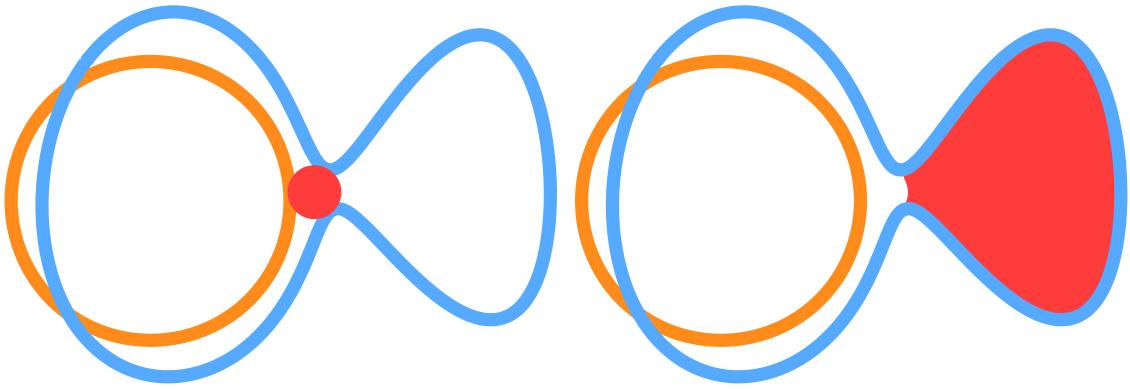


Figure 42: The *ball-distance* (red disk) can produce undesirable results in extreme incompatible cases (left). The *ball-error* measure handles this case by measuring the area of the incompatible feature (right) produced by *relative blending*.

ball-distance may not give an adequate sense of how different two shapes are, as shown in Figure 42-left. In such cases, we propose a new measure, *ball-error*, which takes into account the area in 2D or volume in 3D of incompatible features which are determined by computing a *relative blending* on the shapes A and B. The total *ball-error* is computed as $E_B(A,B) = Area(A - R_B(A)) + Area(B - R_A(B))$ as shown in Figure 42. The discrepancy of a single point on A or B may also be described as the shortest path through the *moat* X to B or A, respectively, similar to the convexity measure in [70]. They measure convexity of a point $\mathbf{s}$ on a closed curve S as the length of the shortest path which does not intersect iS from $\mathbf{s}$ to the convex hull of S.

6.2 Discrepancy Exaggeration

Local discrepancies may be exaggerated by uniformly scaling the radii of all maximal balls in X and reconstructing one curve from the other using the *ball offset* (Fig. 43), as described in Chapter 3. This exaggeration approach proves useful when comparing two shapes or when interactively editing one shape manually so that it perfectly aligns with another shape. In manufacturing, parts are created as copies of an ideal model. Using the exaggeration technique, errors may be visualized quickly by a human

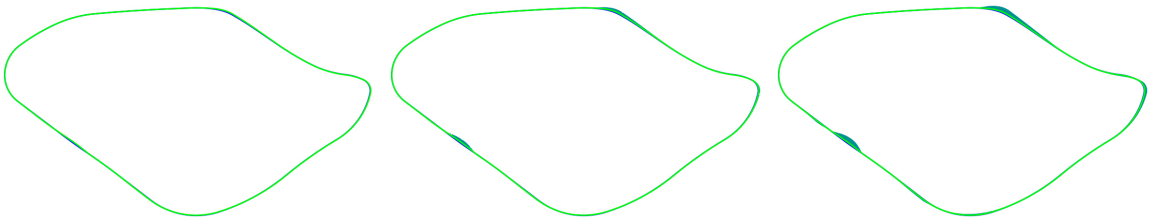


Figure 43: The barely noticeable discrepancy between the original blue and green shapes (left) may be exaggerated by expressing B as the *ball offset* of A, by scaling the offset field 4 times (center) or 6 times (right) and shading the *moat* between A and the exaggerated *ball offset* version of B.

operator to determine whether or not the errors are within acceptable tolerances.

CHAPTER VII

BALL MORPH FOR COMPATIBLE SHAPES

A variety of techniques have been proposed for automatically computing a morph between two curves A and B in the plane (see [51] and [3] for examples). In this chapter, we present a new family of morphs, which we call the *b-morphs*. In Chapter 8, we discuss two related issues: (1) How to compare different morphing solutions and (2) How do the *b-morphs* introduced here compare to other approaches.

7.1 Contributions

We propose a family of three new morphing techniques (that we call *b-morphs*) for which the correspondence and the vertex trajectories are both derived from the maximal disks and their tangential contact points with the curves, as established using the *ball-map* described in Chapter 5.

7.2 Limitation

Our *b-morph* constructions assume that the two curves have been registered and are sufficiently similar, or *b-compatible*. Loosely speaking, our compatibility conditions require that each maximal disk [108] in the finite region bounded by the union of the two curves, or the *moat* X, have exactly one contact point with each curve, as discussed more formally in Chapter 3

Where A and B are similar but not properly registered, one may consider combining a *b-morph* with the animation of a rigid or non-rigid registration [120] or of a smooth space warp [11], as was done for image morphs [12]. Numerous solutions to the automatic registration problem have been proposed using ICP [13],

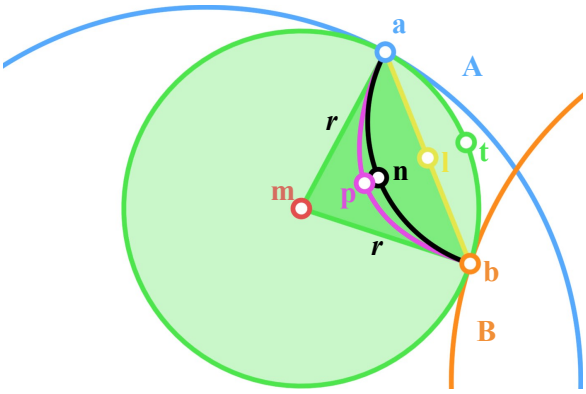


Figure 44: To obtain the point **b** on B that corresponds, through the ball-map, to point **a** on A, we compute the smallest positive r such that $\mathbf{m} = \mathbf{a} + r\widehat{\mathbf{N}}_A(\mathbf{b})$ is at distance r from B and return its closest projection **b** on B. Point **m** is on the medial axis M (red) and defines the center of the circle tangent at both **a** and **b**. The *Circular* (black) and *Parabolic* (purple) *b-morph* trajectories are defined by the inscribing isosceles triangle $\triangle$**amb**. The *Linear b-morph* trajectory is the line segment **ab**.

automatically identified landmarks [80] [49] [83], or distortion minimizing parameterization [121] [102].

7.2.1 B-morph trajectories

For each maximal disk, we consider five *paths* (curve segments) from **a** to **b** (Fig 44):

***Hat*:** The broken line segment from **a** to **m** to **b** (Fig 44 green).

***Linear*:** The straight line segment from **a** to **b** (Fig 44 yellow).

***Tangent*:** The shorter of the two circular arc segments of the boundary of the ball $\mathbb{B}(\mathbf{m}, r)$ that joins **a** and **b** (Fig 44 green).

***Circular*:** The circular arc segment that is orthogonal to A at **a** and to B at **b** (Fig 44 black).

***Parabolic*:** The parabolic arc segment that is orthogonal to A at **a** and to B at **b** (Fig 44 violet).

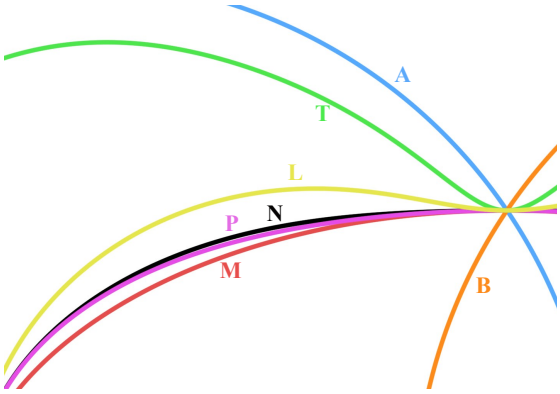


Figure 45: The associated average curves for the various *b-morph* constructions.

The *Circular* and *Parabolic* paths are trivially defined by their enclosing isosceles triangle ($\triangle$**amb**). The *Parabolic* path is the quadratic Bézier curve with control vertices **a**, **m** and **b** and the center of the circle supporting the *Circular* path is the intersection of the tangent to A at **a** and the tangent to B at **b**.

All paths, including the *Linear* path, are symmetric in that the angles where they meet A and B are equal. Swapping the role of A and B does not affect these segments. Hence, the *b-morphs* derived here are *symmetric* and may be inverted easily by swapping the role of A and B.

Let **l** be the midpoint of the *Linear* path and let L be the set of all points **l**. L is the midpoint locus proposed by Asada and Brady [6]. Let **t** be the midpoint of the *tangent* path and T be the set of all points **t**. T is the Process Inferring Symmetry Axis proposed by Layton [68] as a variation of the medial axis. Let **n** be the midpoint of the *Circular* path and N the set of all points **n**. Let **p** be the midpoint of the *Parabolic* path (quadratic B-spline) and P be the set of all points **p**. The construction of these 4 points, along with **m** is illustrated in Fig. 44.

The curves M, L, T, N and P usually differ from one another, but may all be viewed as *averages* of A and B. They are shown superimposed in Fig. 45.

7.3 Examples

A *b-morph* advances with time, each point **a** according to uniform arc-length parameterization along one of the five aforementioned paths. A result for the *Circular b-morph* is shown in Figures 46, 47, and 48 using seven inbetween frames. Note that the curves in Figure 46 are intentionally not aligned and the *b-morph* doesn't produce the rigid body motion one might expect when morphing between two identical shapes. The intersection points of the curves remain fixed throughout the morph. In Figure 47, notice that the inbetweens are not as smooth as the input curves. This is due to the local nature of the *b-morph* which knows nothing of global smoothness. Additional examples of the *Circular b-morph* as well as the *Parabolic* and *Linear b-morphs* can be found in Chapter 8. The future work of combining *b-morphs* with affine motions is discussed in Chapter 13.

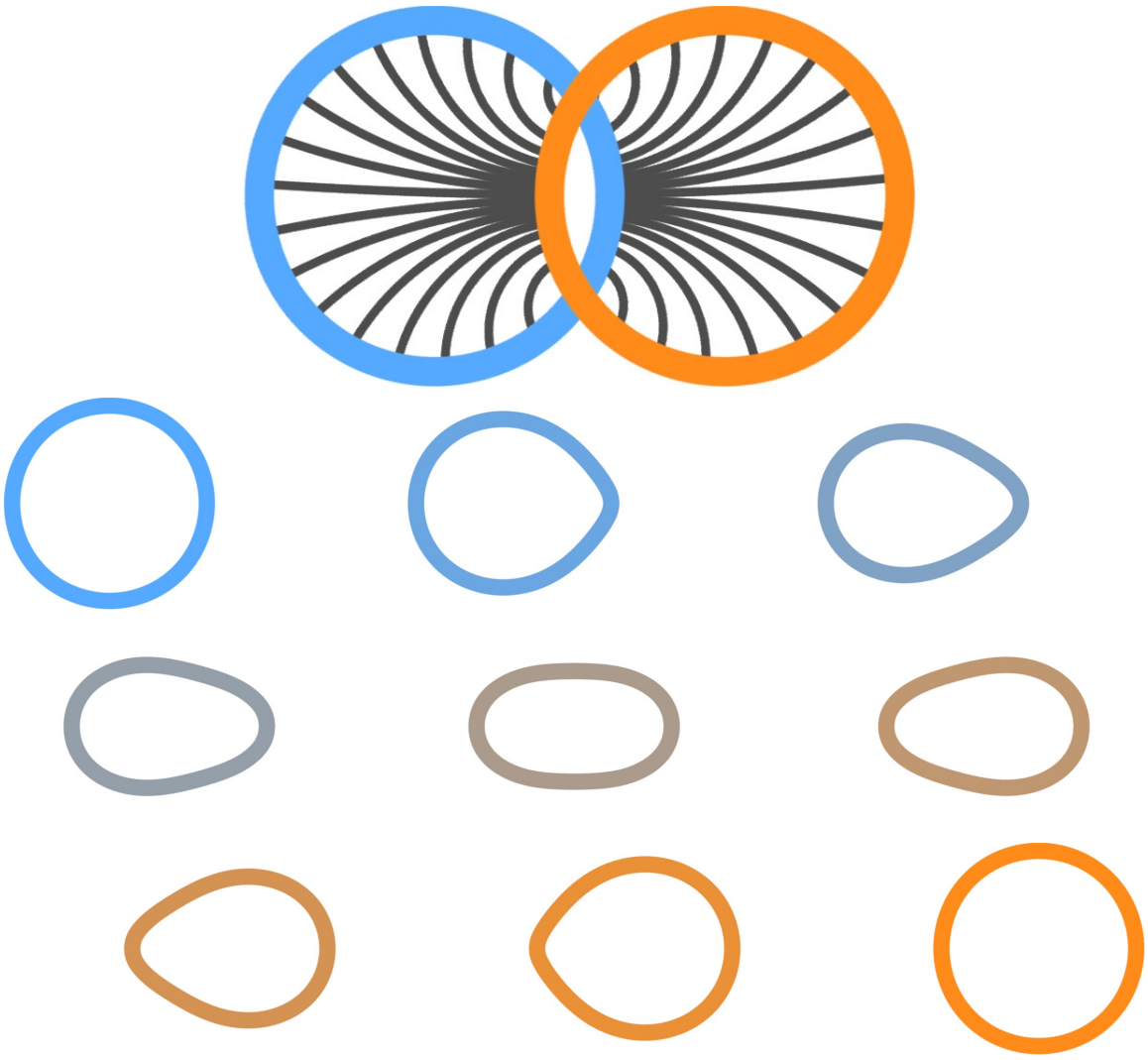


Figure 46: A morph between two offset circles along *Circular b-morph* trajectories (top). Frames are displayed in order from left to right, top to bottom.

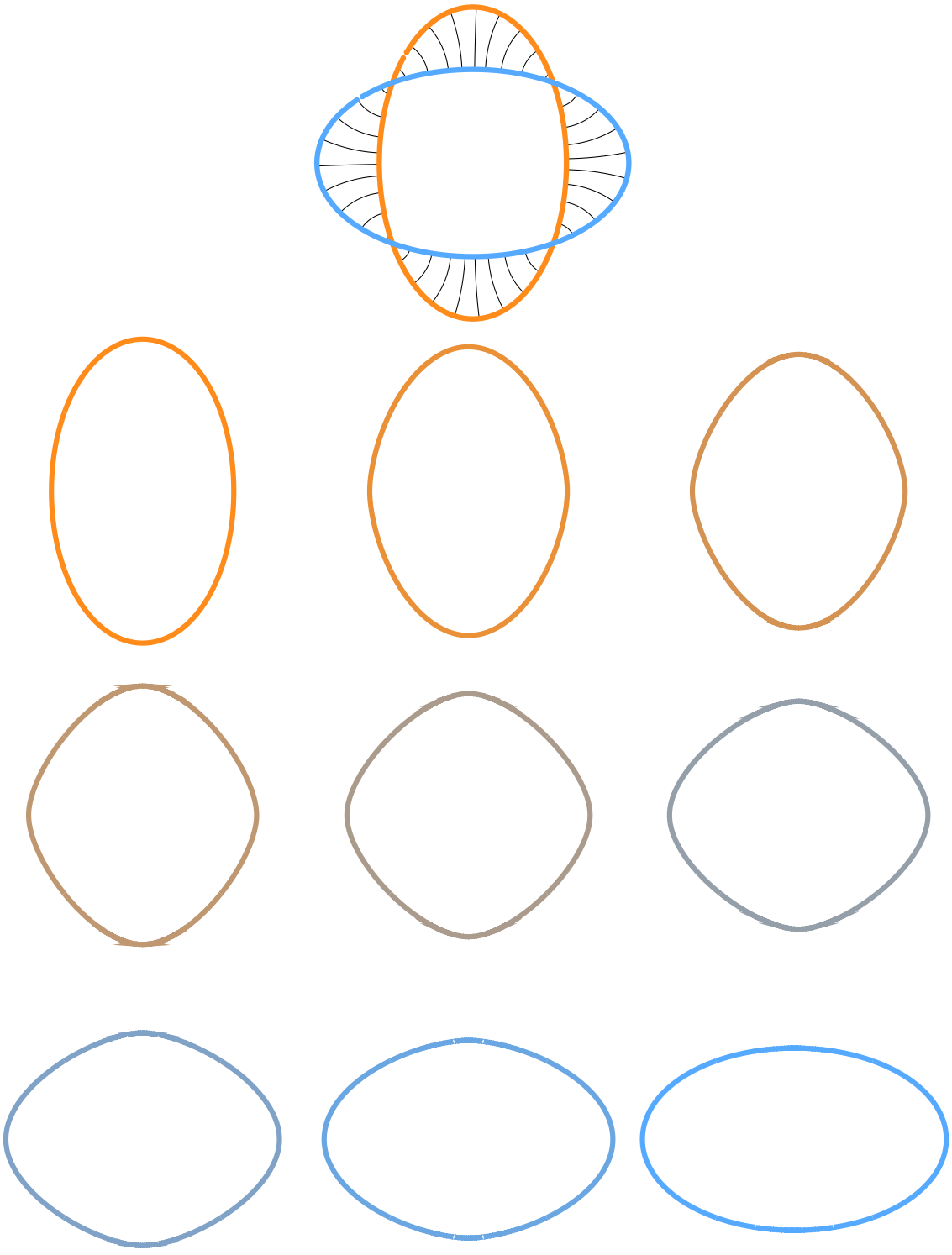


Figure 47: A morph between two ellipses along *Circular b-morph* trajectories (top). Frames are displayed in order from left to right, top to bottom.

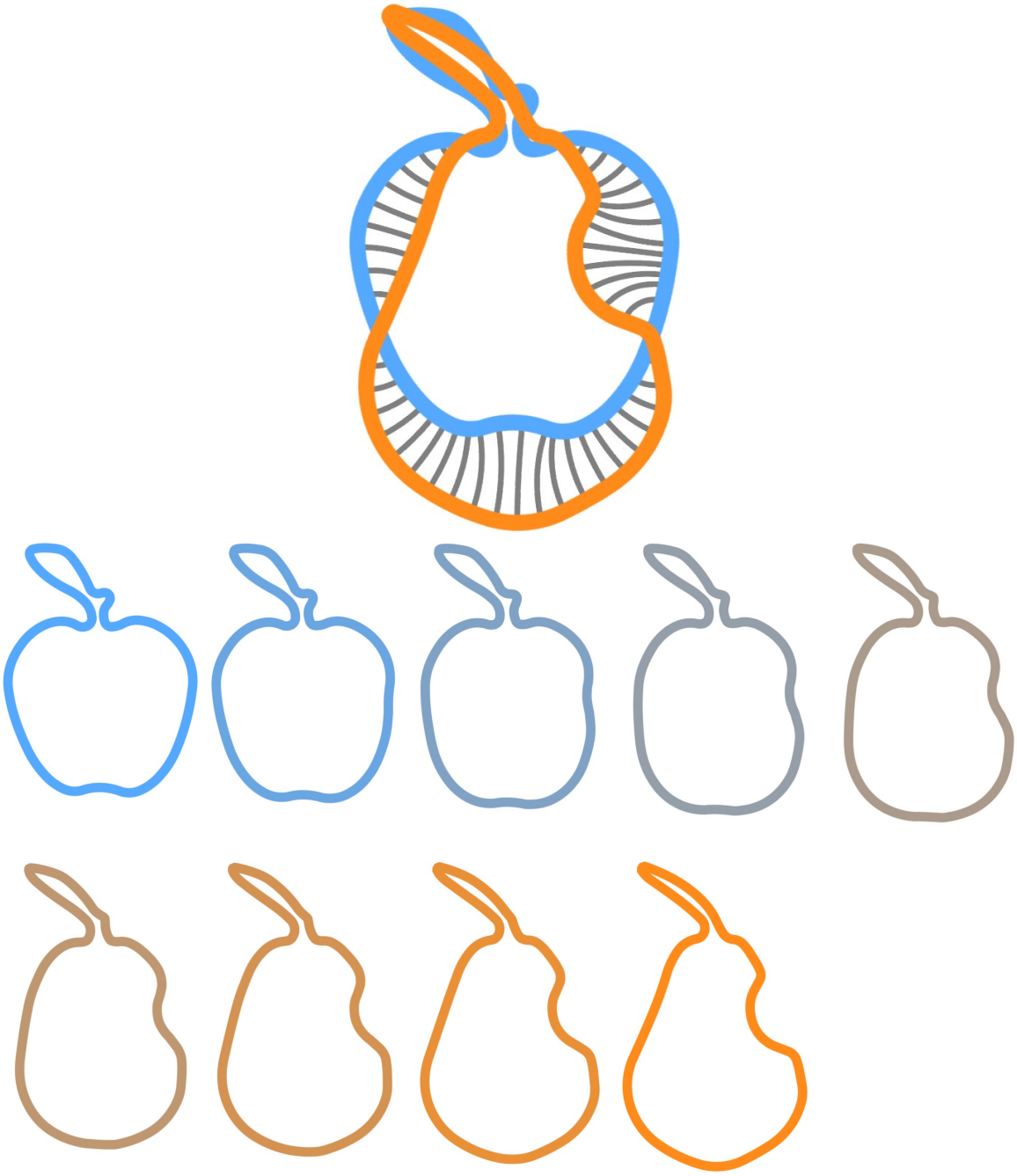


Figure 48: A morph between an apple and pear along *Circular b-morph* trajectories (top). Frames are displayed in order from left to right, top to bottom.

CHAPTER VIII

COMPARISON OF VARIOUS MORPHS

8.1 Prior art on planar curve morphing

A large variety of techniques have been investigated for the automatic generation of in-betweening frames or animations that morph between two planar curves.

We only discuss here techniques that are appropriate to the tight in-betweening problem discussed in Chapter 13. We do not discuss the problems of registration or landmark (salient feature) identification and assume that the curves have been registered.

First, we consider techniques that assume that the correspondence between vertices or samples on both curves is either given by the artist or computed automatically using uniform geodesic sampling, minimization of area or travel [47], curvature-sensitive sample [36], or optimization of matching to affine transformations extracted from an example morph [110].

If the correspondence is given, the simplest approach is to use a linear interpolation between corresponding pairs. Linear trajectories are computed between these pairs of points on A and B in order to produce morph curves with vertices $\mathbf{v}_i = \mathbf{a}_i + t(\mathbf{b}_i - \mathbf{a}_i)$. We include this *Linear Interpolation* (LI) in our benchmark set of approaches that we compare to the *b-morphs*. This naïve approach may lead to unpleasant artifacts, such as self-intersections in the intermediate frames (as for example pointed out by [104]). The *Linear Interpolation* fails to take into account the relative orientation and curvature of the curves at the corresponding points.

To take these into account, a popular morphing technique proposed by [103] for polygonal curves interpolates the lengths of corresponding edges and the angles at

corresponding vertices and uses optimization to ensure that the curve closes properly. We include a simple version of this approach, which we call *Curvature Interpolation* (*CI*) in our benchmark set. When it is applied to open curve segments, we ensure that the interpolating frames meet at end-point constraints by retrofitting them through a trivial similarity transformation (rotation, scaling, and translation).

A different approach that takes into account the relative orientation and curvature of the two curves at the corresponding samples is to compute the local coordinates of each vertex in the coordinate system defined by its neighbors on each curve. Then, the corresponding local coordinates are averaged linearly to produce a *desired* set of local coordinates for a given frame. Iterative techniques may be used to construct a curve that satisfies the two endpoint constraints and minimizes the discrepancy between the actual and *desired* local coordinates. Variations of these techniques have been successfully used [2][131][46]. We include a simple version of this approach, which we call *Laplace Blending* (*LB*), in our benchmark set.

Vertex trajectories and correspondences may also be obtained by solving a PDE or by computing a gradient field that interpolates the two contours and then following the steepest gradient to obtain the trajectory of each point or equivalently [133], the in-between frames may be obtained as iso-contours of that field. A heat propagation formulation may be used to characterize the desired field [33]. We include a simple version of this approach, which we call *Heat Propagation* (*HP*), in our benchmark set.

Several approaches for morphing closed curves use compatible triangulations [5] of their interior [123][52][3] or compatible skeletons to ensure rigidity [106][26]. We don't consider these since they aren't applicable to our problem of open curves. Other approaches blend distance fields to both surfaces [64][32].

We separate the approaches that establish correspondence using a direct geometric criterion (as opposed to a global optimization or feature recognition as discussed

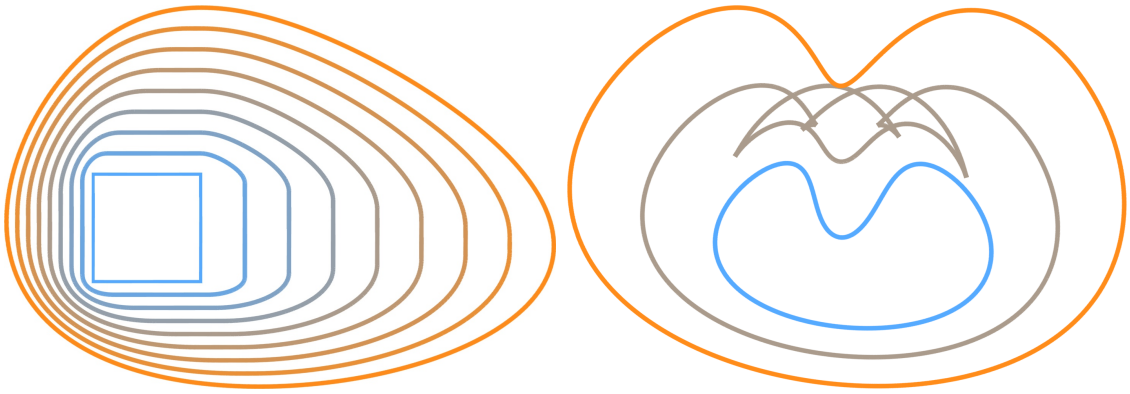


Figure 49: Example Minkowski morphs between convex (left) and non-convex (right) shapes.

above) into three categories: (1) Proximity-based, (2) Orientation-based, and (3) both Proximity- and Orientation-based.

The popular distance-based approach is the closest point projection, which to each point **a** on A maps a point **b** on B that minimizes the distance to **a**. Variations of this approach are used for Iterative Closest Point (ICP) registration [13]. We do not include this approach, the *Closest Projection* (*CP*) morph, because of the limitation of the *c-compatibility* constraint, as defined in Chapter 3. None of the examples used in our benchmark set are *c-compatible*.

An orientation-based approach is the Minkowski morph [98], which yields satisfying results for convex shapes, even when the shapes are not aligned (see Fig. 49 left). For smooth curves, the approach establishes a correspondence between points with the same normal. Unfortunately, as shown in Fig. 49 (right), the approach may yield self-intersecting frames when the two curves are not convex. Hence, we do not include it in our benchmark.

Our *b-morphs* are both proximity- and orientation-based in that correspondence is determined locally and based on a ball that is tangent to corresponding points. The *Heat Propagation* morph is also proximity- and orientation-based since the trajectories are used to identify correspondence and are orthogonal to each of the input curves.

8.2 Measures

Comparing morphs is difficult since criteria used in the comparison may be application-dependent and subjective. Nevertheless, we propose here quantitative measures of the quality of a morph. We propose seven measures of quality inadequacy (i.e. lack of quality): *travel distance*, *distortion*, *stretch*, *local acceleration*, *surface area*, *average curvature*, and *maximal curvature*. We compare the three *b-morphs* to each other and also four simple morphing techniques which we have implemented (*Linear Interpolation* (*LI*), *Curvature Interpolation* (*CI*), *Laplace Blending* (*LB*), *Heat Propagation* (*HP*)). We first discuss how we sample space and time. Then, we provide details of the measures used here to compare morphs.

8.2.1 Measure normalization

Three of the studied morphs (*Linear Interpolation*, *Curvature Interpolation*, and *Laplace Blending*) assume a given correspondence. For simplicity, we use a uniform arc-length sampling to produce the same number of uniformly distributed samples on each curve. The three *b-morphs* use the ball-map correspondence. The other morphs compute their own correspondence. This sampling disparity makes it difficult to compute measures for a fair comparison.

Consider for example the problem of measuring the average *travel distance*. This should be the integral of travel distances. The problem is how to fairly select the integration element. If for example we use the *CP* morph, then the average distance measured for a set of uniformly distributed samples will depend on whether we start from A or B. The average *travel distance* is a property of the mapping, and should be dependent on the sampling. A measure that so blatantly depends on the sampling is clearly incorrect.

To overcome this problem, each reported measure is the average of two measures, one computed by sampling A and one computed by sampling B. For the first measure,

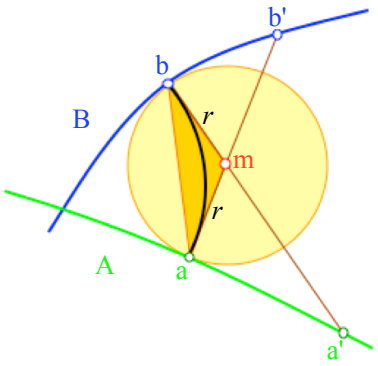


Figure 50: Comparison of the travel distances of the *b-morph* trajectory and the two *CP* morph trajectories that correspond to any point **m** of the medial axis of X.

we sample the departure curve A using a dense set of samples that are uniformly distributed on each curve so as to be separated by a prescribed geodesic distance u. For each sample $\mathbf{a}_i$ on A, we compute the corresponding point $\mathbf{b}_i$ on the arrival curve B so that $\mathbf{b}_i$ is the image of $\mathbf{a}_i$ by the mapping associated with the particular morphing scheme. We compute a measure m_i associated with the trajectory from $\mathbf{a}_i$ to $\mathbf{b}_i$ and the associated weight $w_i = (dist(\mathbf{a}_{i-1}, \mathbf{a}_i) + dist(\mathbf{b}_{i-1}, \mathbf{b}_i) + dist(\mathbf{a}_i, \mathbf{a}_{i+1}) + dist(\mathbf{b}_i, \mathbf{b}_{i+1}))/4$. Then, we report the normalized weighted average $(\Sigma w_i m_i)/(\Sigma w_i)$. For the second measure, we sample the arrival curve B as before using the same geodesic distance u. For each sample $\mathbf{b}_i$ on B, we compute the corresponding point $\mathbf{a}_i$ on the departure curve B, so that $\mathbf{b}_i$ is the image of $\mathbf{a}_i$ by the mapping associated with the particular morphing scheme. Then, we proceed as above and report the average of the two.

8.2.2 Measures used in our comparison

We have implemented the following seven measures of morph quality.

Travel distance For each sample $\mathbf{a}_i$, we measure m_i as the arc length of the trajectory to the corresponding point $\mathbf{b}_i$. Then, as explained in Section 8.2.1, we report the weighted average E_{travel} of these from A to B and vice-versa.

Although we don't use the *CP* morph in our benchmark comparison, we show that the travel distance of the *CP* morph is never less than that of any of the three *b-morphs*. Consider an arbitrary point on the medial axis $\mathbf{m} \in M$ of the *moat* X and its closest projections $\mathbf{a}$ on A and $\mathbf{b}$ on B, as shown in Figure 50. Now let $\mathbf{b}'$ be the pre-image of $\mathbf{a}$ by the *Closest Projection* from B to A, and let $\mathbf{a}'$ be the pre-image of $\mathbf{b}$ by the *Closest Projection* from A to B. With each point $\mathbf{m} \in M$, we associate three travel distances: the travel distance $|\mathbf{b} - \mathbf{a}'|$, the travel distance $|\mathbf{a} - \mathbf{b}'|$, and the travel distance of the *ball-map* from $\mathbf{a}$ to $\mathbf{b}$, which is the length of the circular-arc trajectory from $\mathbf{a}$ to $\mathbf{b}$ and never exceeds $|\mathbf{a} - \mathbf{m}| + |\mathbf{m} - \mathbf{b}| = 2r$. Because the corresponding ball fits in the *moat* X, $|\mathbf{m} - \mathbf{b}'|$ and $|\mathbf{m} - \mathbf{a}'|$ cannot be smaller than r.

Travel distance can be a useful measure in a variety of applications, such as finding an efficient route for an advancing army on the battlefield which minimizes travel, thereby conserving fuel, time and every soldier's energy.

Stretch We define stretch as the average of the integral over time of the stretch factor for an infinitesimal portion of the curve. We compute its discrete approximation as follows. Let $\mathbf{a}$ and $\mathbf{a}'$ be consecutive samples on A. Let $L(\mathbf{a}, t)$ be the length of the segment of $A(t)$ between $\mathbf{a}(t)$ and $\mathbf{a}'(t)$. We compute stretch for a sample $\mathbf{a}_i \in A$ as

$$s_i = \sum_{t \in [0, 1-\epsilon]} (|L(\mathbf{a}, t + \epsilon) - L(\mathbf{a}, t)|)$$

The weighted average $E_{stretch}$ is then reported as described above.

The measure of stretch can be important when computing the range of flexibility of an elastic material. If the material experiences a stretch during its range of motion that is beyond its prescribed maximum tolerance, failure may occur.

Acceleration *Acceleration*, or unsteadiness [120], is defined as the derivative of the expression of velocity in the local, time-evolving frame, and measures the lack of

steadiness of the motion.

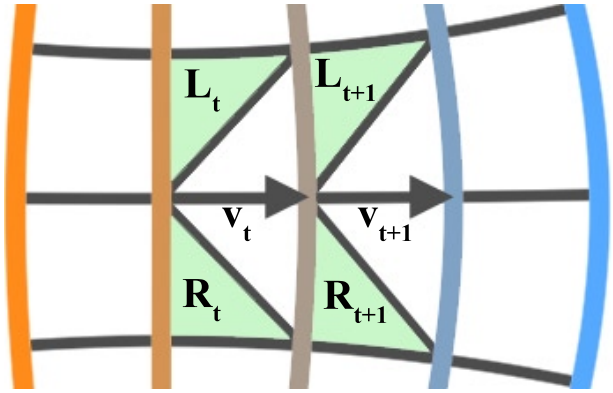


Figure 51: Computation of *acceleration* (steadiness) for a given vector $\mathbf{v}$ of the morph trajectory is computed relative to the neighboring triangles (green).

To compute $E_{acceleration}$, let $\mathbf{a}_t$ denote the position of a sample $\mathbf{a}$ at a time t. We approximate the instantaneous velocity of $\mathbf{a}_t$ by the vector $\overrightarrow{\mathbf{a}_t\mathbf{a}_{t+\epsilon}}$. For each such velocity on a morph trajectory, we compute two barycentric coordinate vectors $\overrightarrow{B}_L(\overrightarrow{\mathbf{a}_t\mathbf{a}_{t+\epsilon}})$ and $\overrightarrow{B}_R(\overrightarrow{\mathbf{a}_t\mathbf{a}_{t+\epsilon}})$ relative to the left and right neighboring triangles L_t and R_t as shown in Fig. 51. The steadiness at a point $\mathbf{a}_t$ is then computed as:

$$\begin{aligned} g_t = \; & \tfrac{1}{2}|\overrightarrow{B}_L(\overrightarrow{\mathbf{a}_{t-\epsilon}\mathbf{a}_t}) - \overrightarrow{B}_R(\overrightarrow{\mathbf{a}_{t-\epsilon}\mathbf{a}_t})| \\ + \; & \tfrac{1}{2}|\overrightarrow{B}_L(\overrightarrow{\mathbf{a}_t\mathbf{a}_{t+\epsilon}}) - \overrightarrow{B}_R(\overrightarrow{\mathbf{a}_t\mathbf{a}_{t+\epsilon}})| \end{aligned}$$

We compute the acceleration measure as the sum of the g_t terms over the trajectory of each point $\mathbf{a}_i$ and report their weighted average $E_{acceleration}$, as described above.

Acceleration is important in detecting instabilities in a motion. For example, consider points as passengers on an airplane, where the plane represents the local frame flying (evolving) along a trajectory. When the plane encounters turbulence, the passengers' coordinates within the plane may not be steady (due to local acceleration), causing discomfort. If the motion is free of acceleration, the passengers will remain perfectly still in the local coordinates defined by the plane.

Distortion At each point along the evolving curve and at each time, the amount of distortion is proportional to $1/cos\theta$, where θ is the angle between the direction of travel and the normal to the evolving curve. If the direction of travel is always in the normal direction, then the morph is said to be free from distortion.

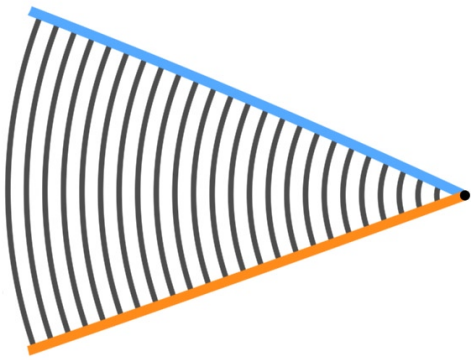


Figure 52: The *b-morph* produces a pure rotation with zero distortion, zero stretch and zero acceleration between linear segments in 2D.

Let $\mathbf{a}$ and $\mathbf{a}'$ be consecutive samples on A and $L(\mathbf{a}, t)$ define the length of the segment $\overrightarrow{\mathbf{aa}'}$. Let $V(\mathbf{a}, t)$ define the length of the segment $\overrightarrow{\mathbf{a_t a_{t+\epsilon}}}$. We compute

$$E_{distortion} = \sum_{t\in[0,1-\epsilon]} \left| 1 - \frac{\frac{1}{2}(L(\mathbf{a},t) + L(\mathbf{a},t+\epsilon)) \cdot \frac{1}{2}(V(\mathbf{a},t) + V(\mathbf{a}',t)}{Area(\mathbf{a_t a'_t a'_{t+\epsilon} a_{t+\epsilon}})} \right|$$

It was shown in [24] that (1) the *b-morph* is a C^{k-1} isotopy when the input curves are C^k for $k \geq 2$ and (2) that the *Circular b-morph* is free from distortion when morphing between line segments (in 2D) of A and B (Fig. 52).

When a surface is reconstructed from a morph with zero distortion, a rectangular texture applied to the surface will retain its angles of $\pi/2$, although there may be stretching along one or both of the axes.

8.2.3 Mesh measures

In addition to our 2D measures, we also present results of 3D measures of surface area and also average and maximum squared mean curvature [78] of the resulting triangle mesh surfaces constructed by interpolating the input curves along the $z - axis$. In

applications of surface reconstruction from 2D planar contours, minimizing the surface area and smoothness of the resulting reconstruction is often desirable (see Fig. 53).

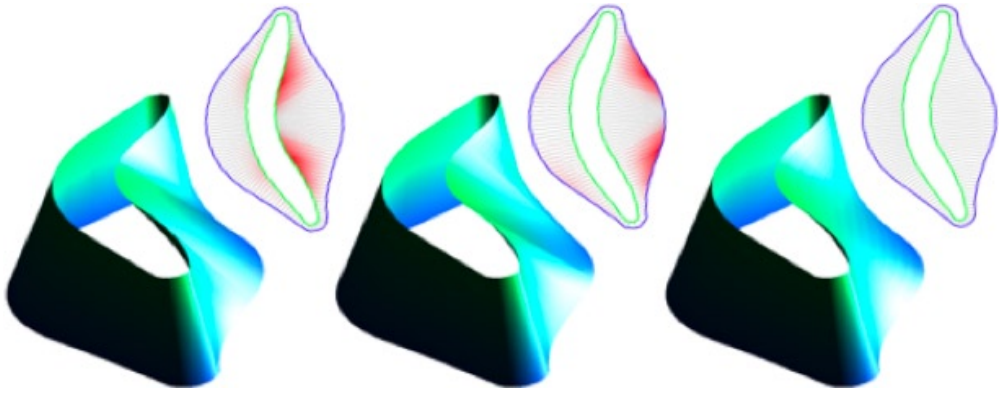


Figure 53: We show the slice-interpolating surface reconstructed using a *Closest Projection* morph from-green-to-blue (left), the reverse *Closest Projection* morph from-blue-to-green (center), and the symmetric *Circular b-morph* (right) which appears smoother. The amount of local *distortion* is shown in red on the 2D drawings. Clearly, the *b-morph* produces less distortion than either of the *CP* morphs. It also produces a shorter average travel distance.

8.3 Results

We first compare the *b-morphs* to our benchmark set using two different test cases, as shown in Figures 8.3 through 69. Then, we compare two of the *b-morphs* to the best two other morphs (Laplace and Heat) on a test case between an apple and a pear. The results are organized by example, showing the trajectories, the surface reconstructions, and 9 frames of each. The frames of animation are ordered from left to right, and top to bottom. At the end of all the morphs for each example is the chart showing the measures, where all results are normalized by dividing by the maximum result for each measure.

The first test case (Figures 8.3-61) shows a morph between two offset circles. In order to simplify the problem of computing correspondences for the methods which rely on uniform arc-length parameterization, we split each curve into 2 open curves at the intersection points.

Our experiments demonstrate that the average *travel distance* is the shortest when using the *Linear b-morph* , as we discussed earlier, and that the *Circular b-morph* has the least amount of distortion. The *HP* morph is the closest in terms of appearance and measure to the *Circular b-morph*.

The second test case (Figures. 8.3-69) shows a set of symmetric 'S' shaped curves. This example highlights the strength of the morphs which compute their own correspondence (*HP*, *b-morphs*). The other results, which define correspondence through uniform arc-length parameterization, exhibit extreme distortion and travel lengths and also produce self-intersections with the original curves. Again, the *HP* morph is closest to the family of *b-morphs* in terms of measure and appearance.

The final test case (Figures 8.3-74) uses contours representing an apple and a pear. We show the best four morphs (*Linear b-morph*, *Circular b-morph*, *Heat Propagation* and *Laplace Blending*) and compare their measures. The measures for these four are similar. Travel distance and distortion are still minimized for the *Linear* and *Circular b-morphs*, respectively. The *LB* approach wins in terms of *acceleration*, *stretch* and *curvature*.

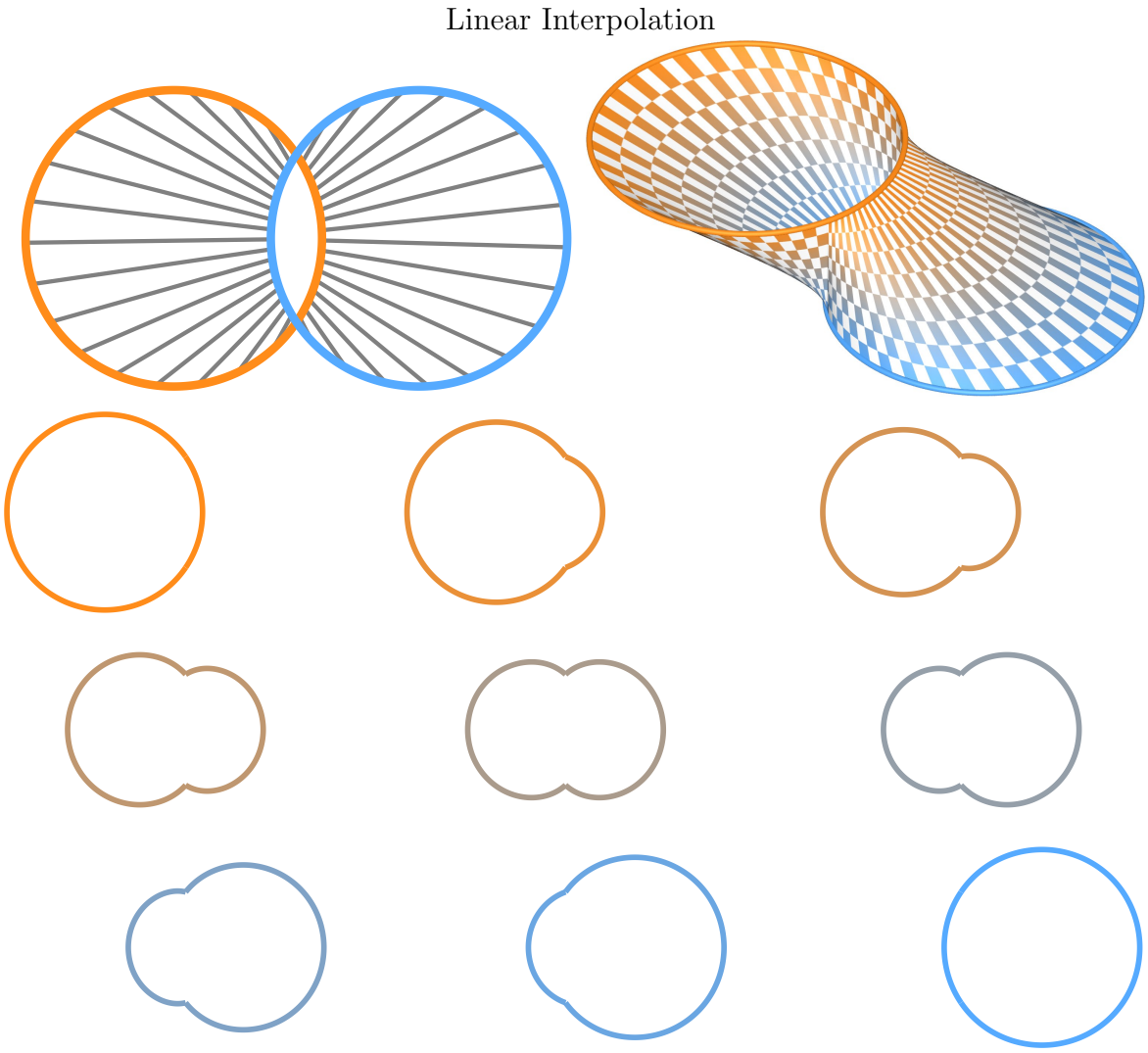


Figure 54: The result of using the Linear Interpolation morphing algorithm

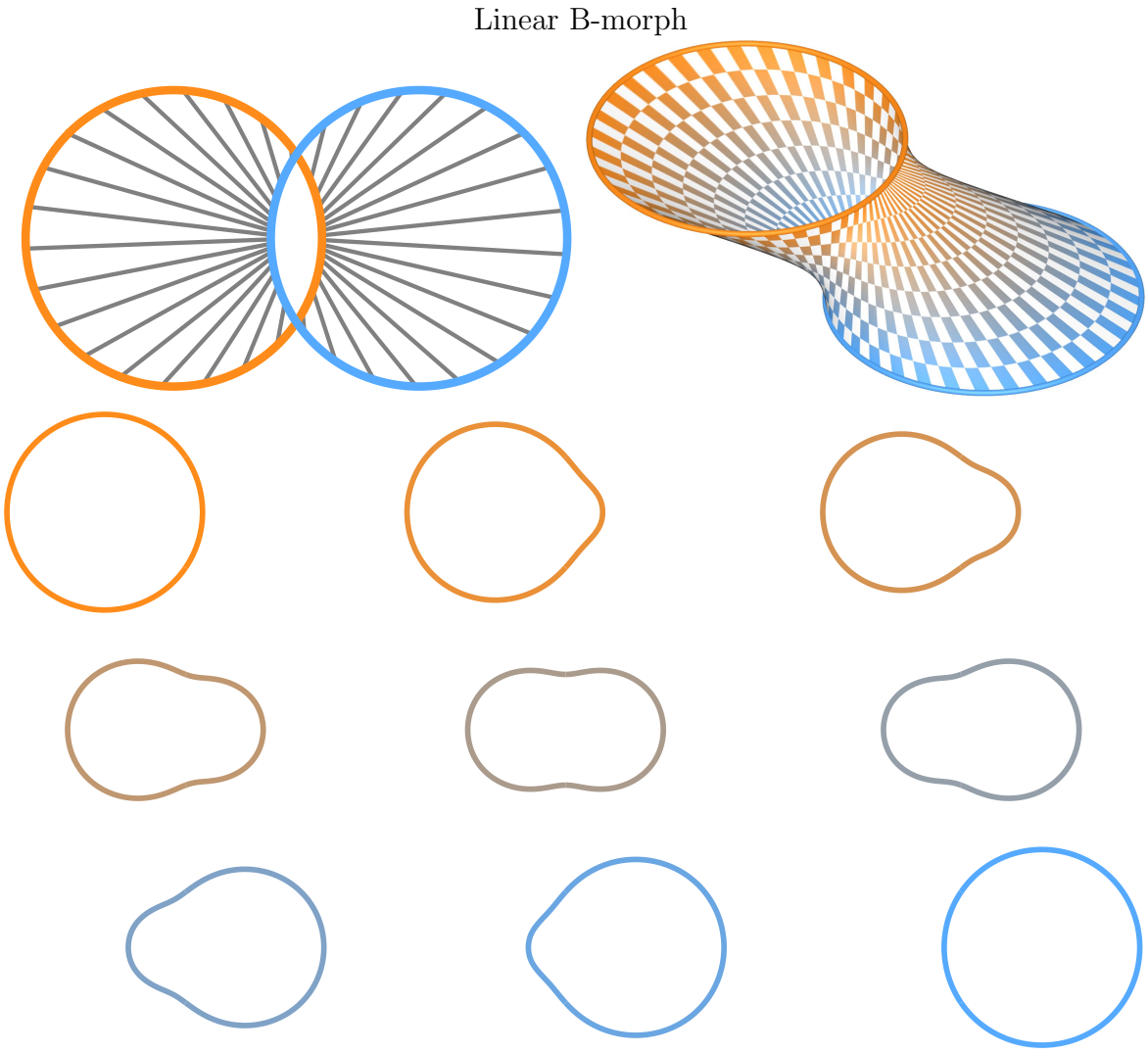


Figure 55: The result of using the Linear B-morph morphing algorithm

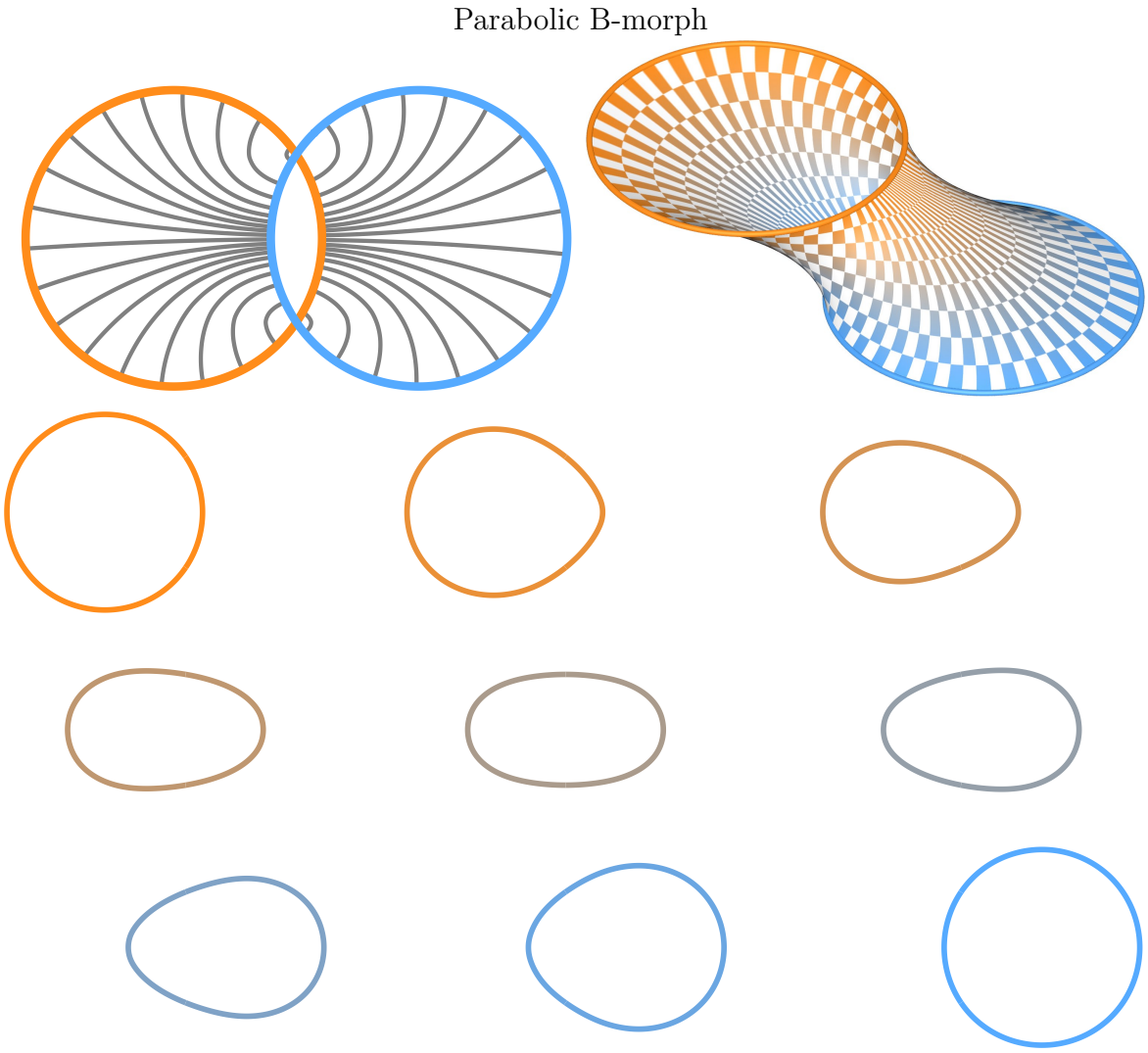


Figure 56: The result of using the Parabolic B-morph morphing algorithm

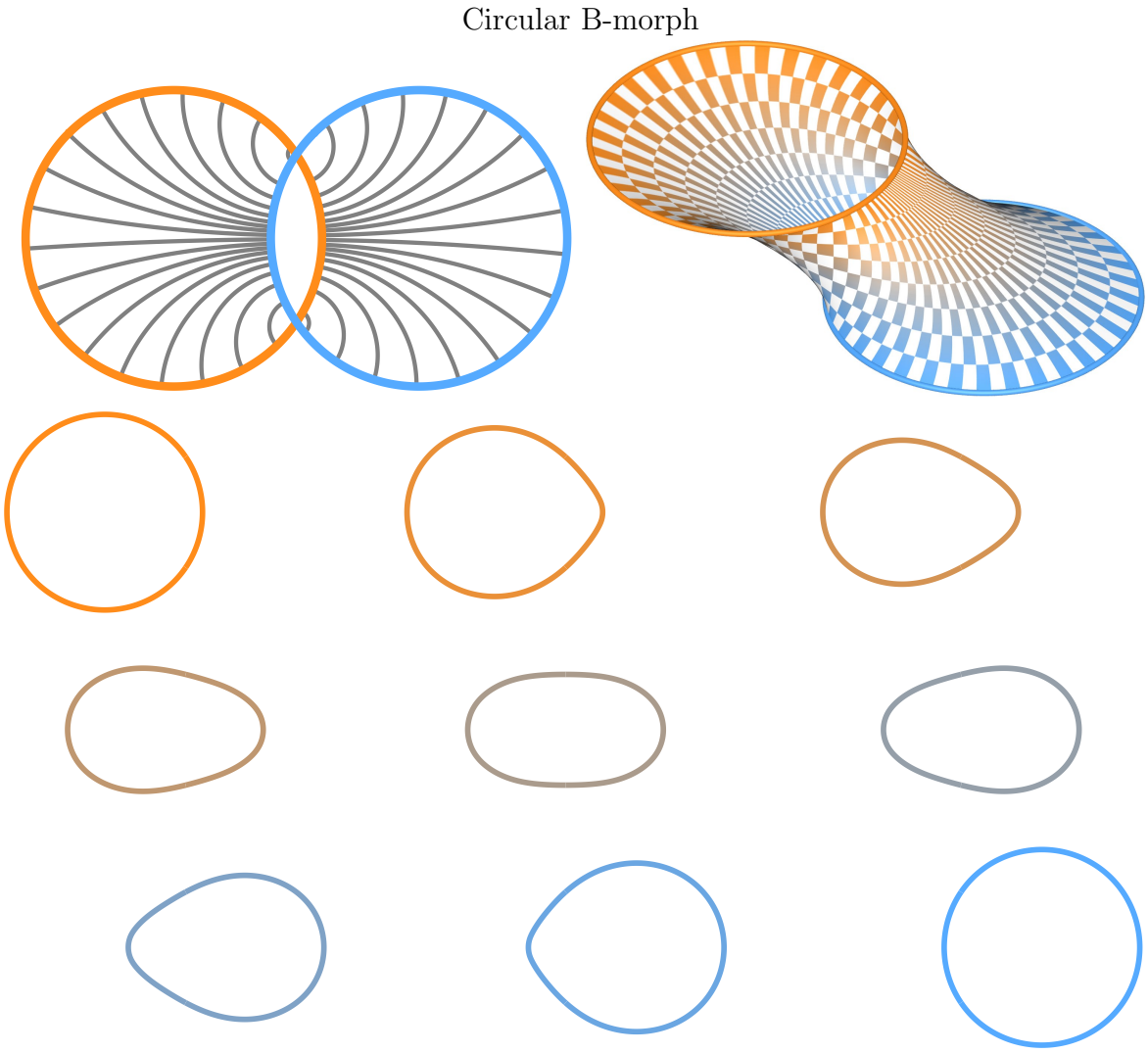


Figure 57: The result of using the Circular B-morph morphing algorithm

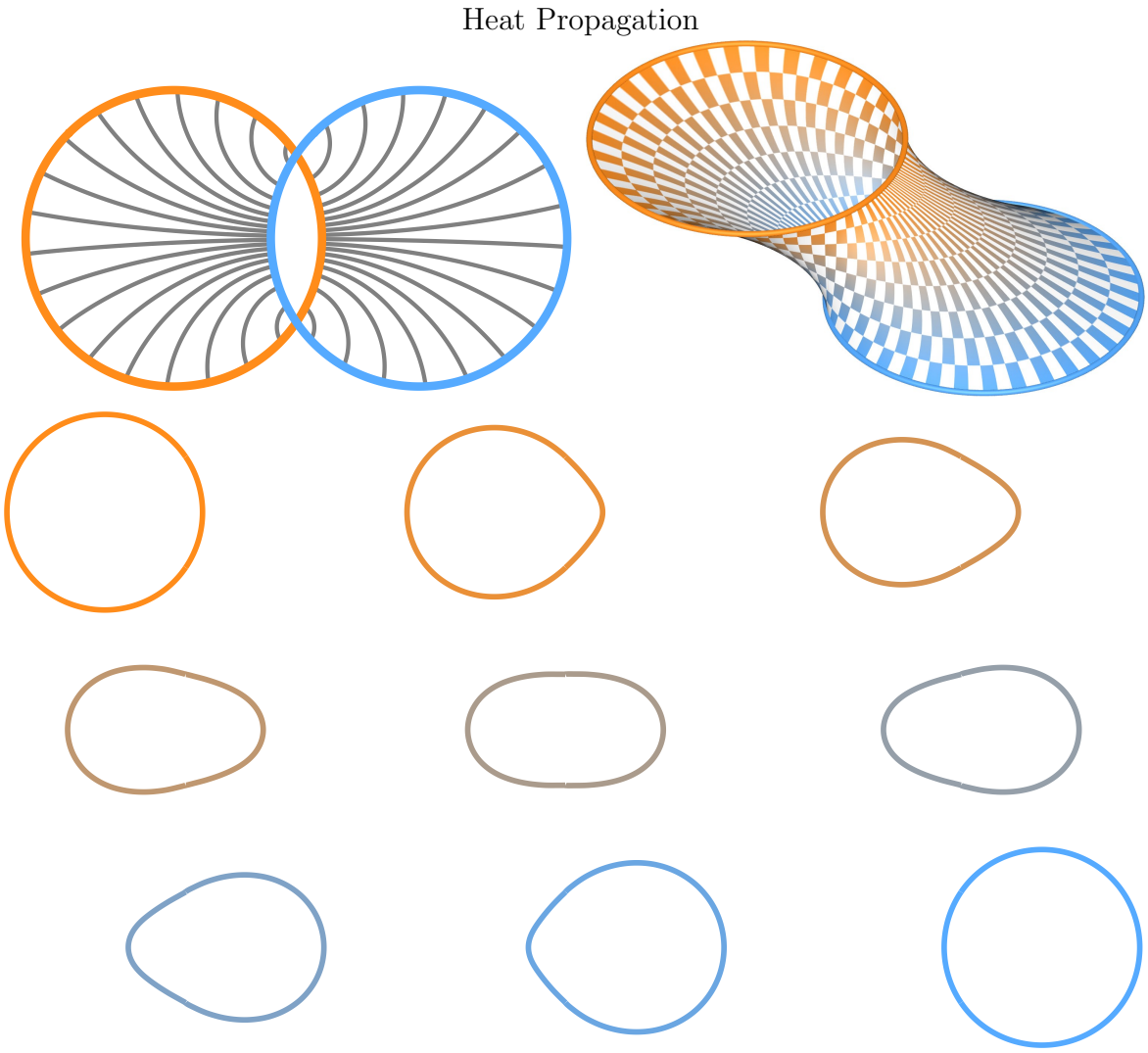


Figure 58: The result of using the Heat Propagation morphing algorithm

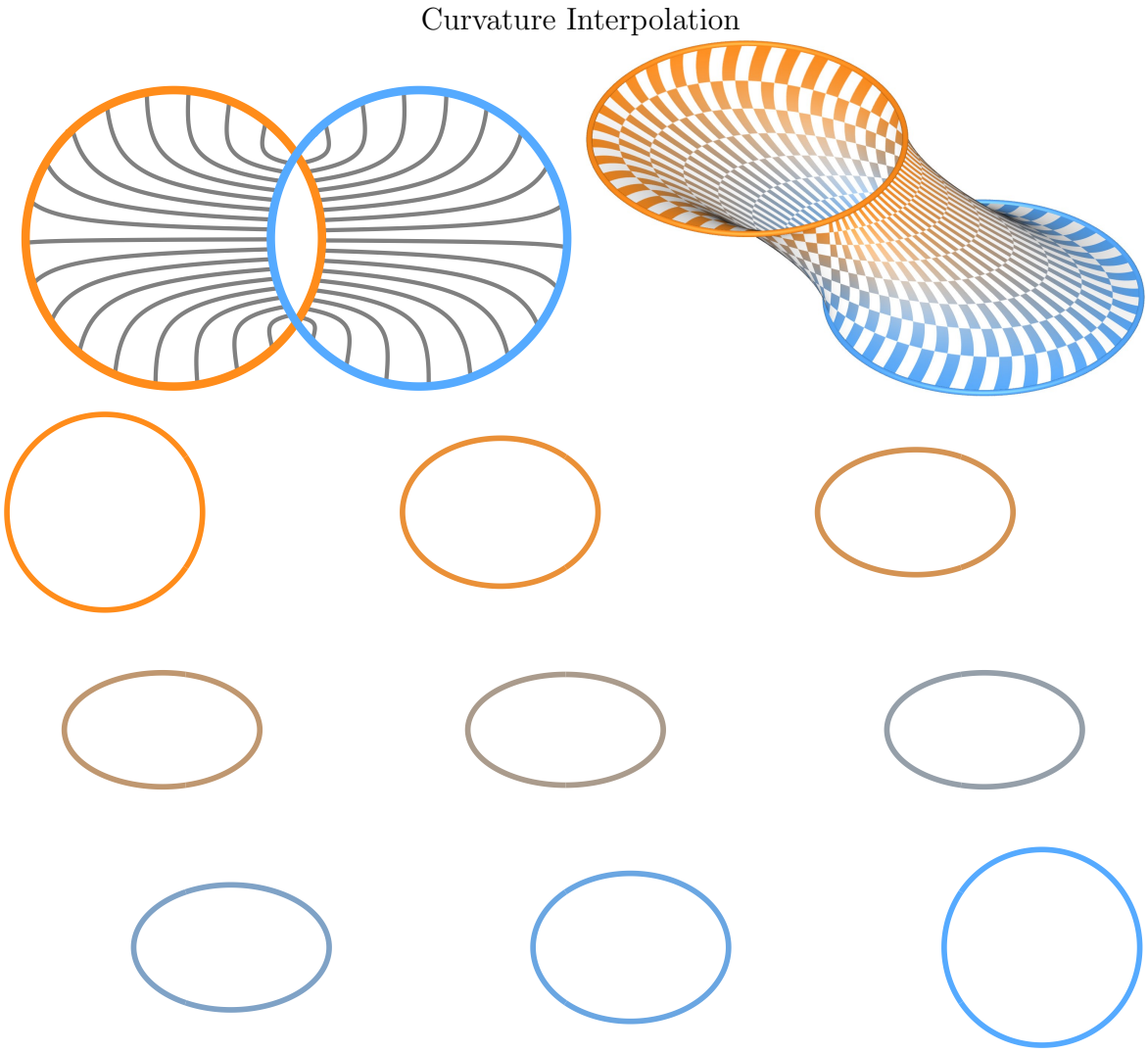


Figure 59: The result of using the Curvature Interpolation morphing algorithm

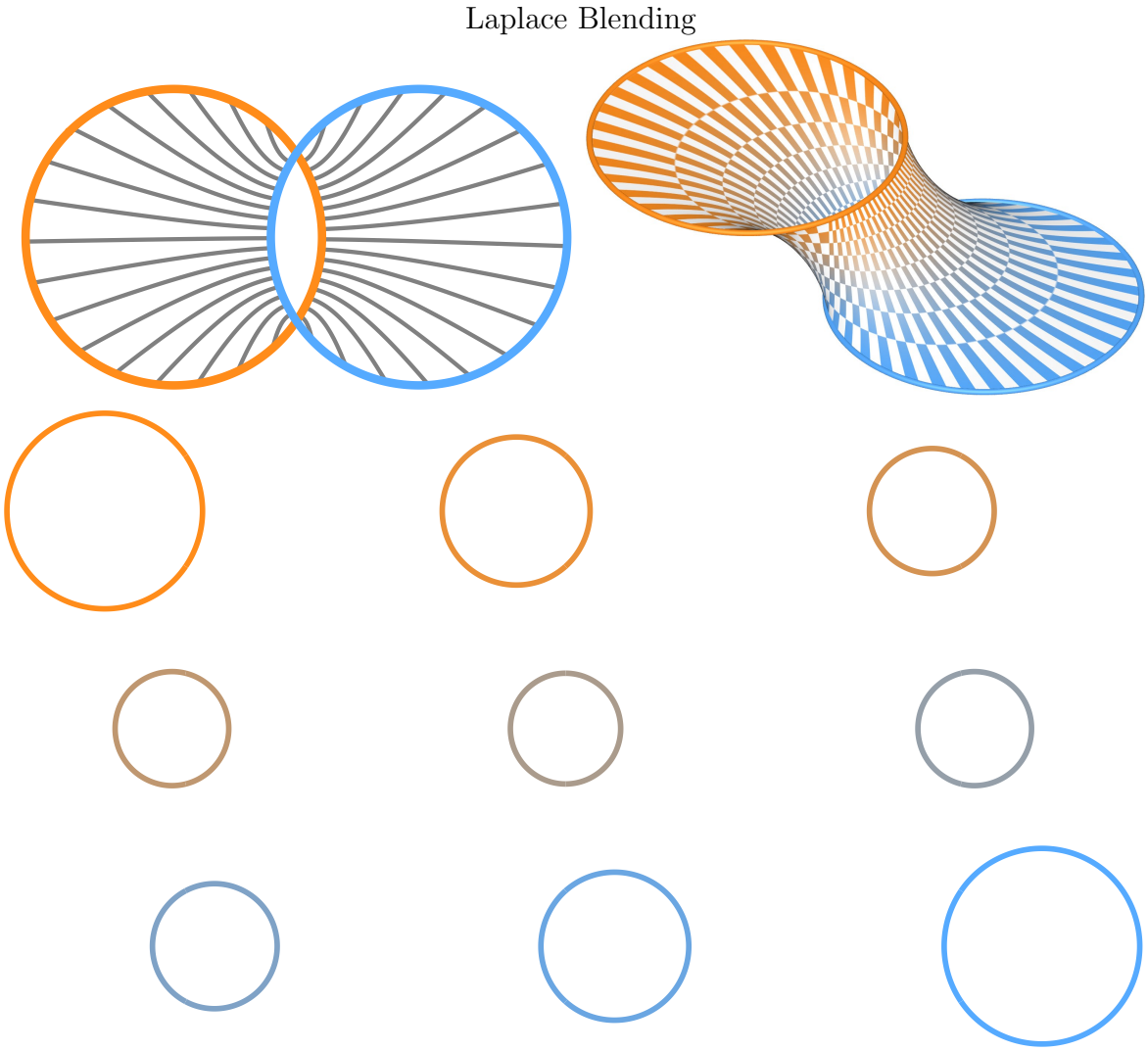


Figure 60: A result of using the Laplace Blending morphing algorithm on a pair of offset circles.

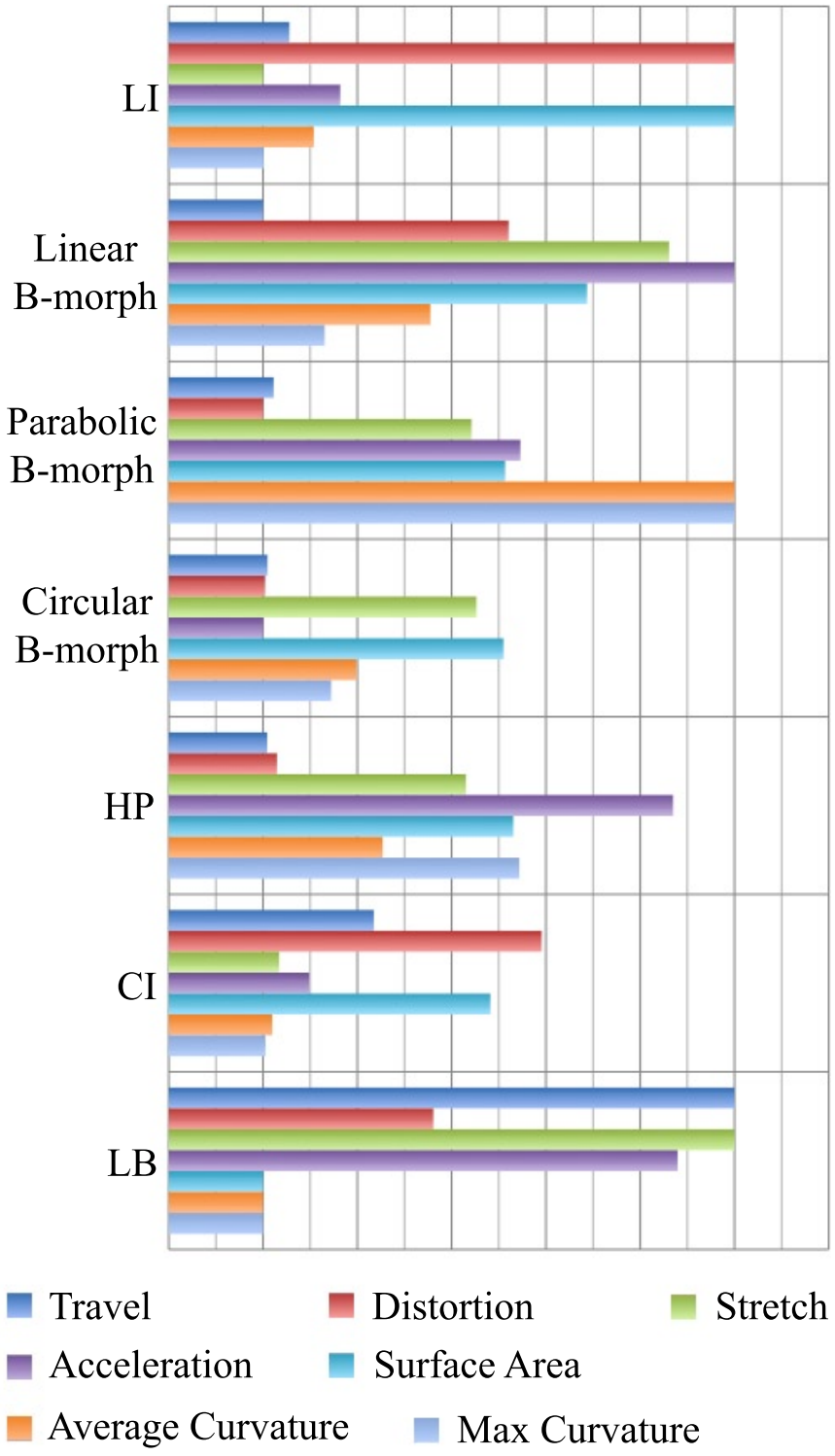


Figure 61: The measures for each morphing algorithm for the pair of offset circles. Each measure is normalized independently by dividing by the maximum result.

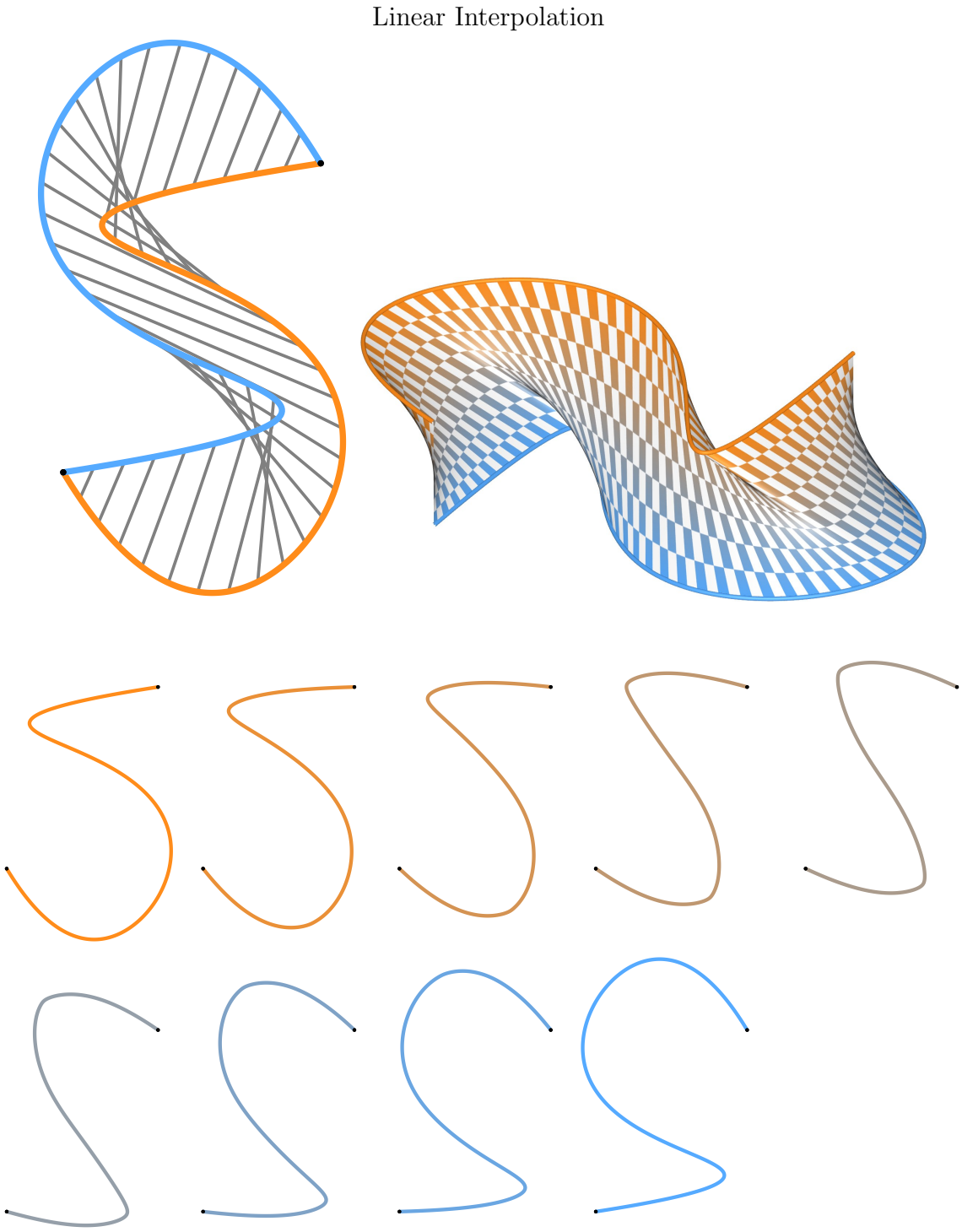


Figure 62: The result of using the Linear Interpolation morphing algorithm

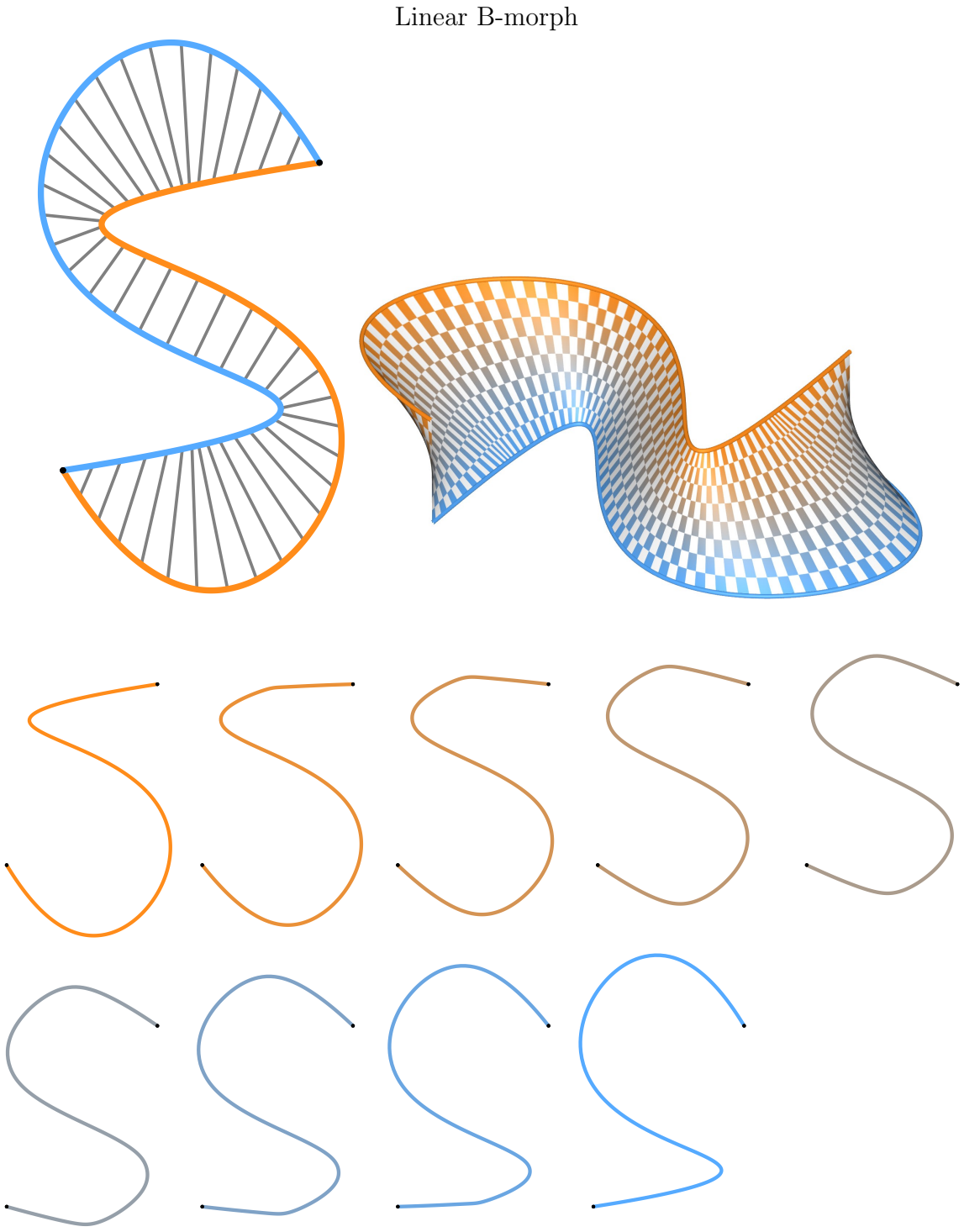


Figure 63: The result of using the Linear B-morph morphing algorithm

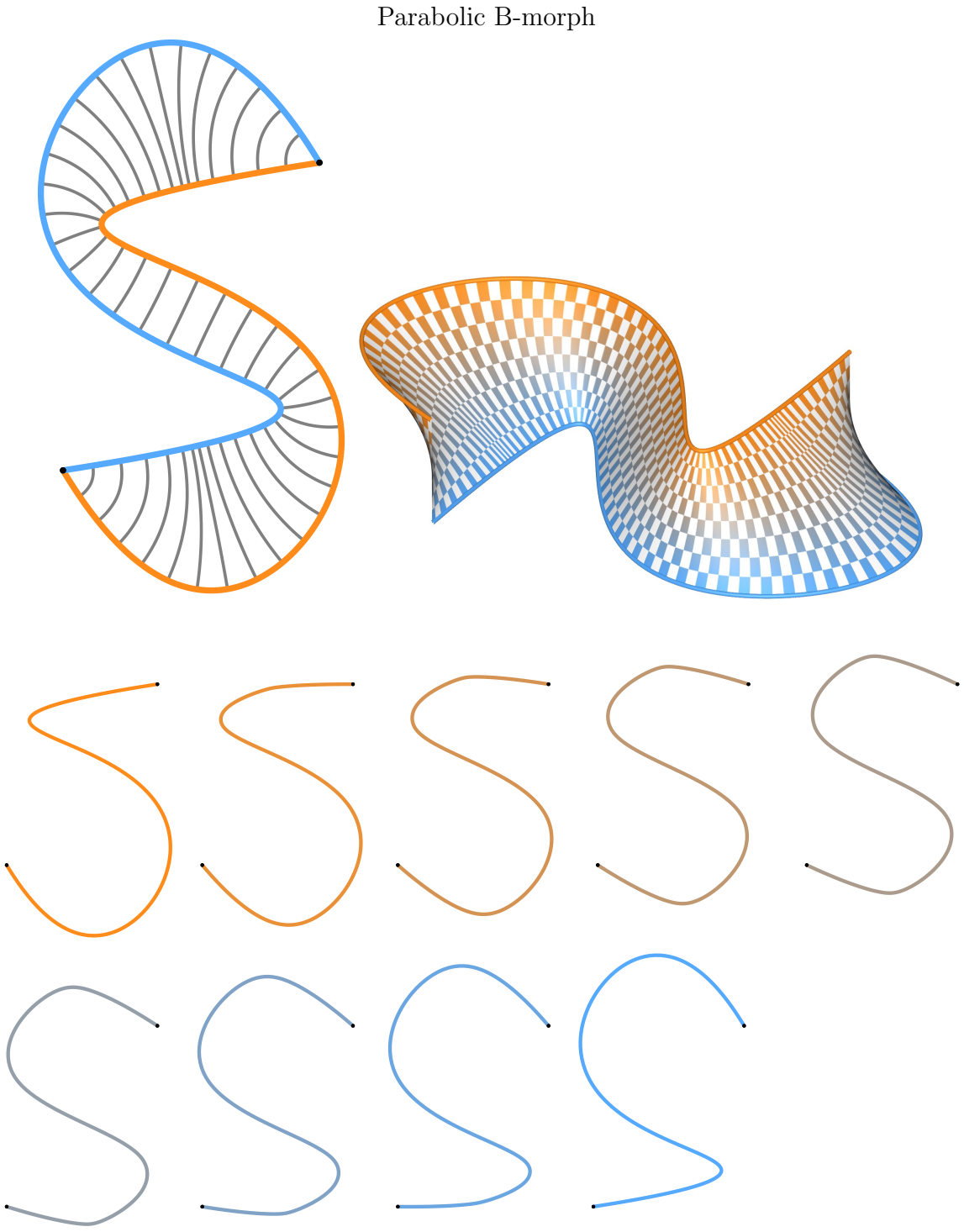


Figure 64: The result of using the Parabolic B-morph morphing algorithm

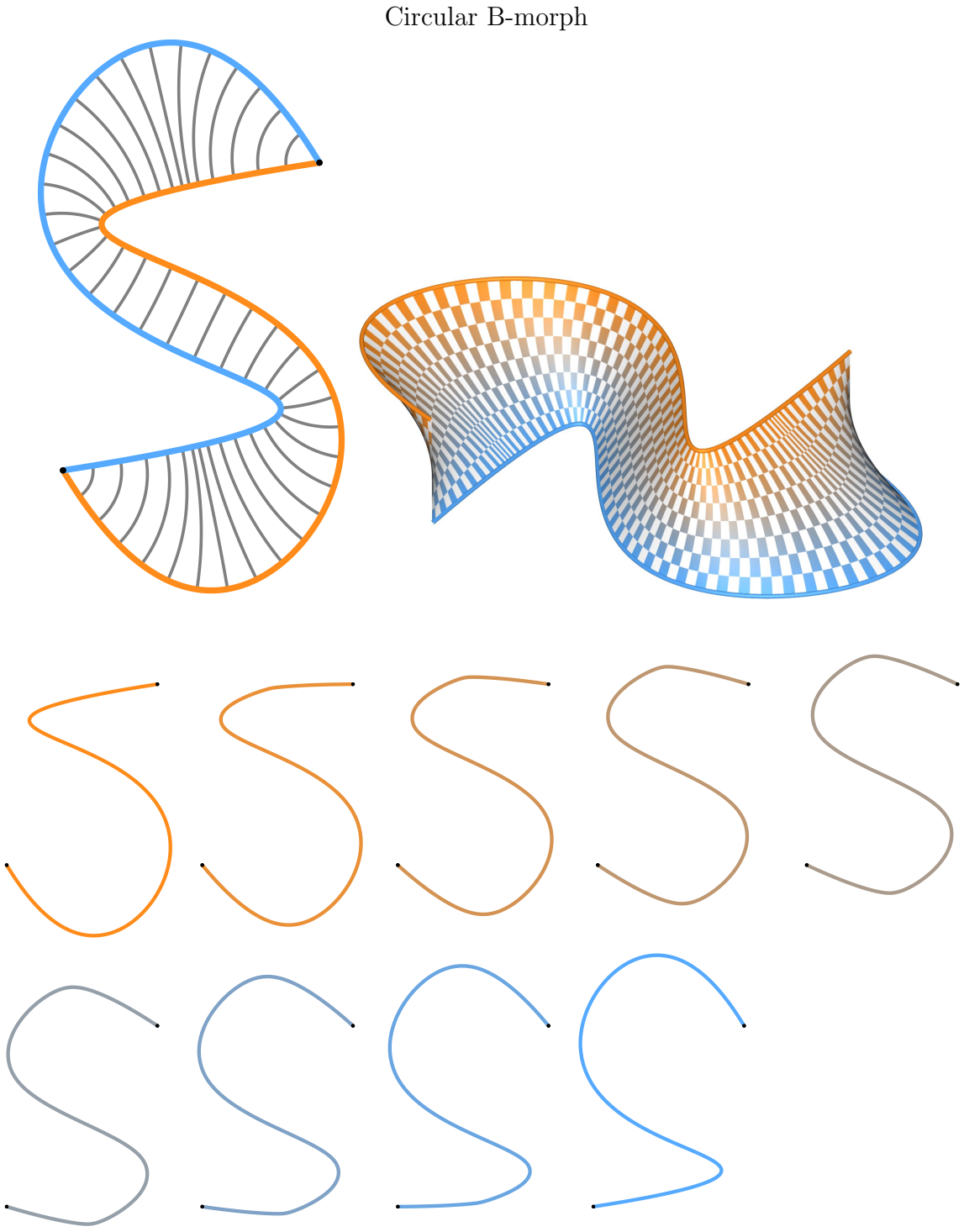


Figure 65: The result of using the Circular B-morph morphing algorithm

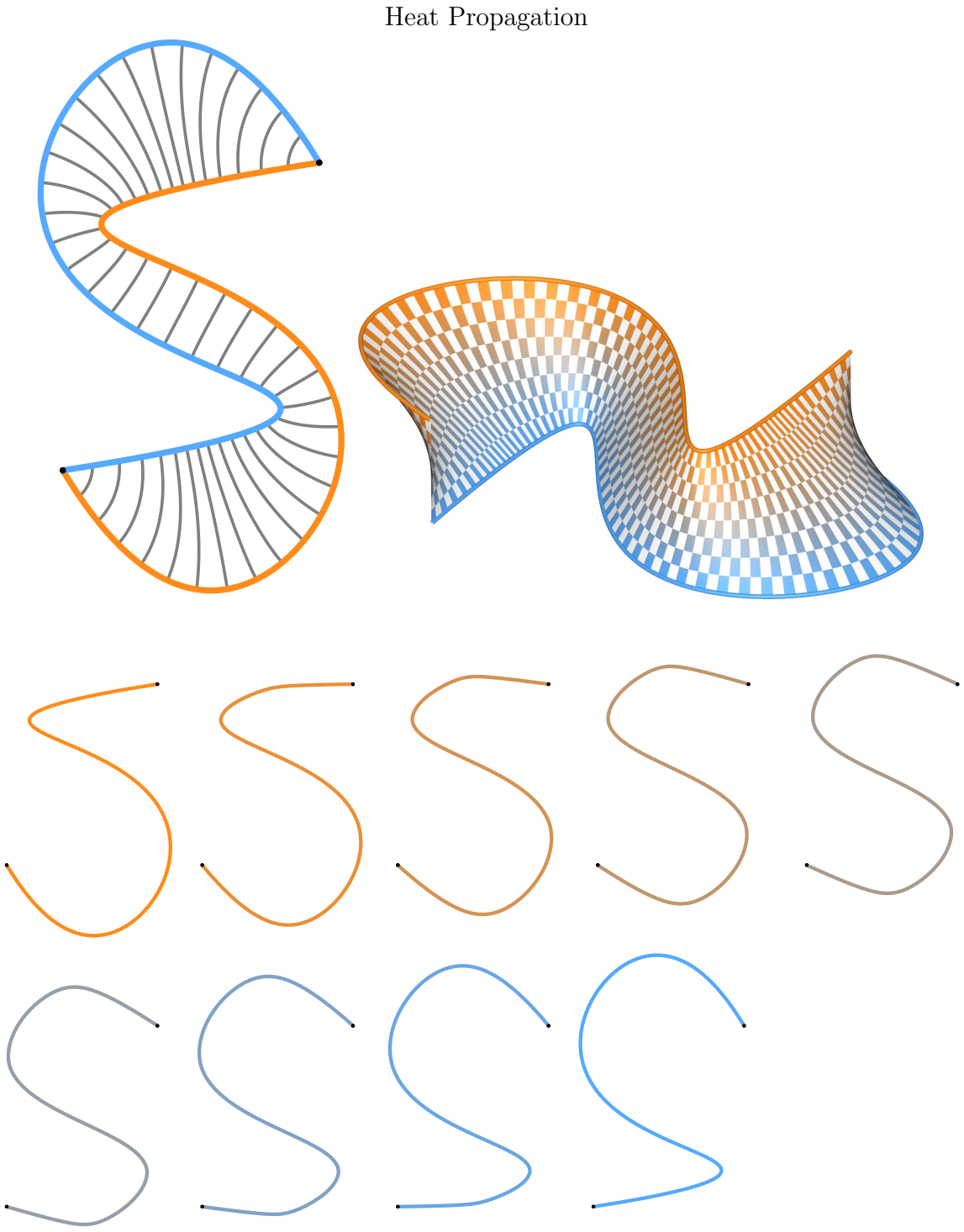


Figure 66: The result of using the Heat Propagation morphing algorithm

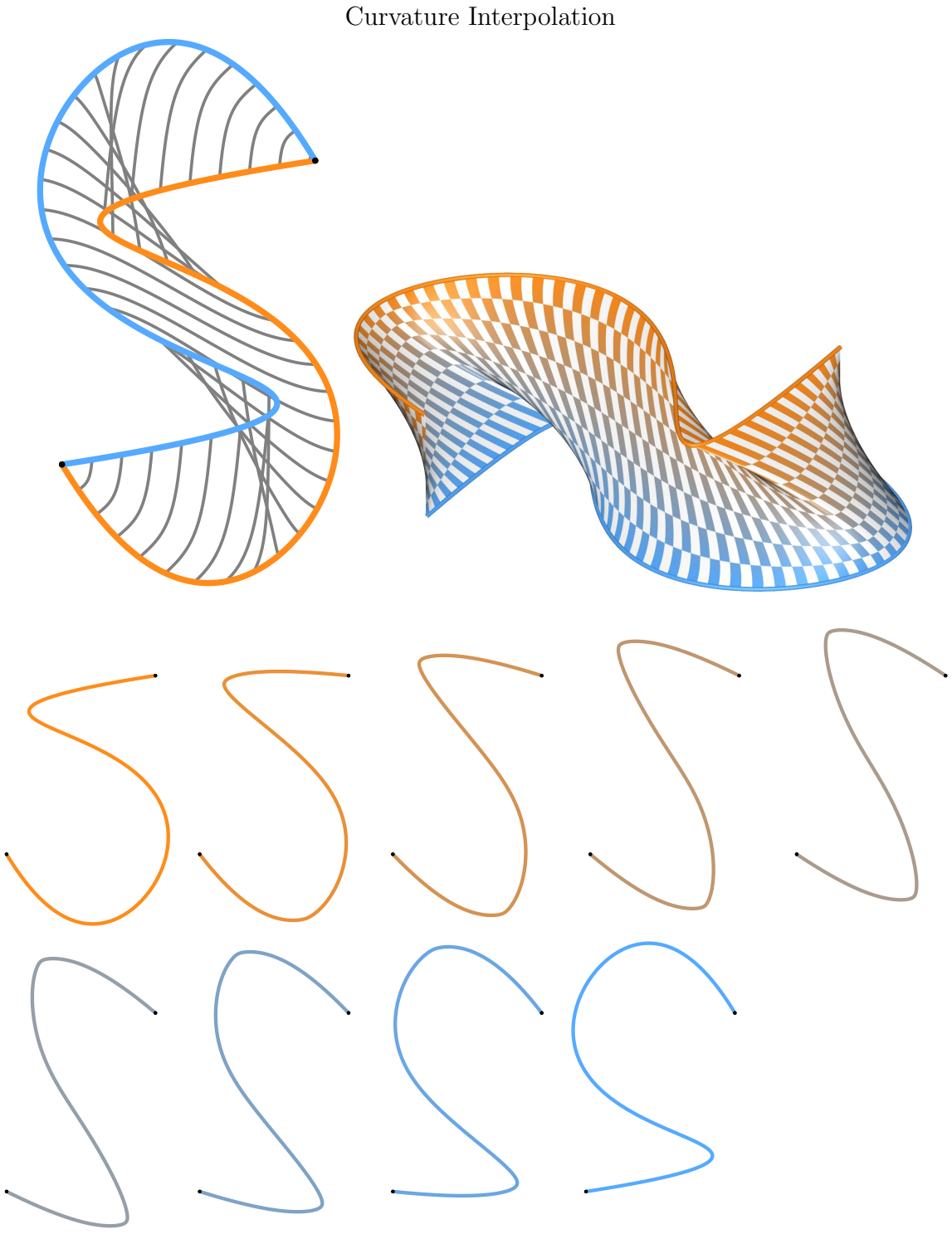


Figure 67: The result of using the Curvature Interpolation morphing algorithm

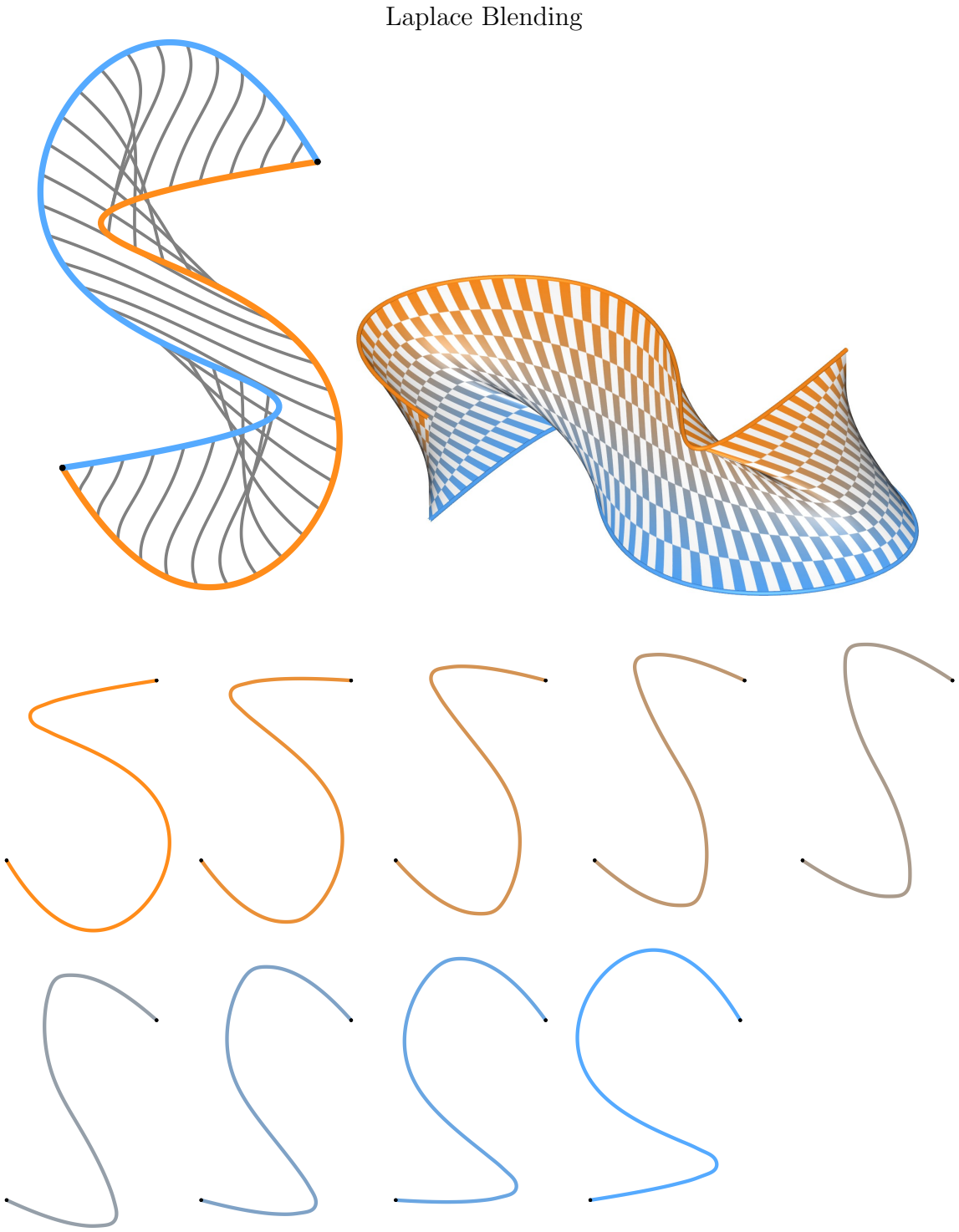


Figure 68: A result of using the Laplace Blending morphing algorithm.

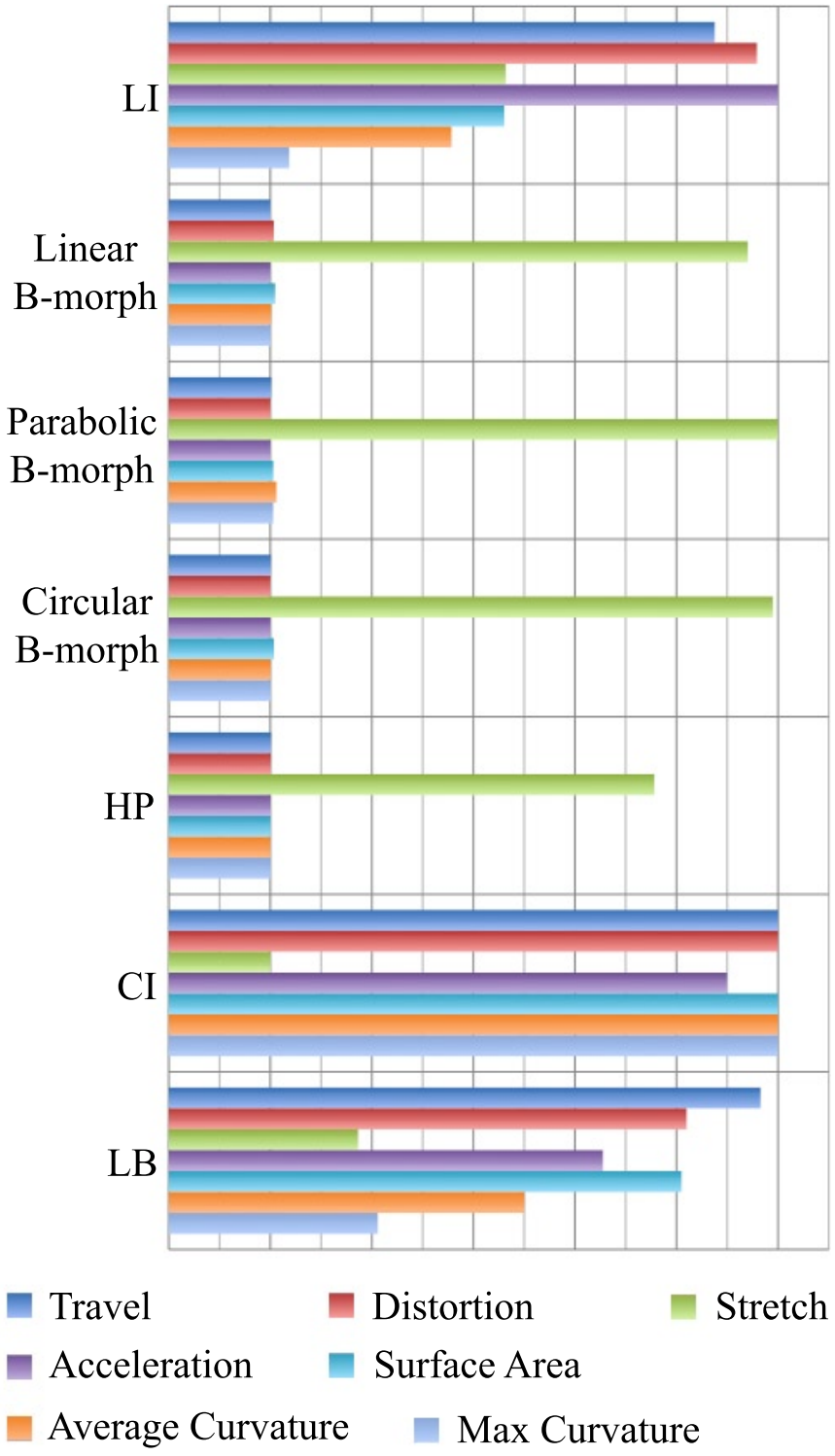


Figure 69: Morph measures for a set of 'S'-shaped curves. Each measure is normalized independently by dividing by the maximum result.

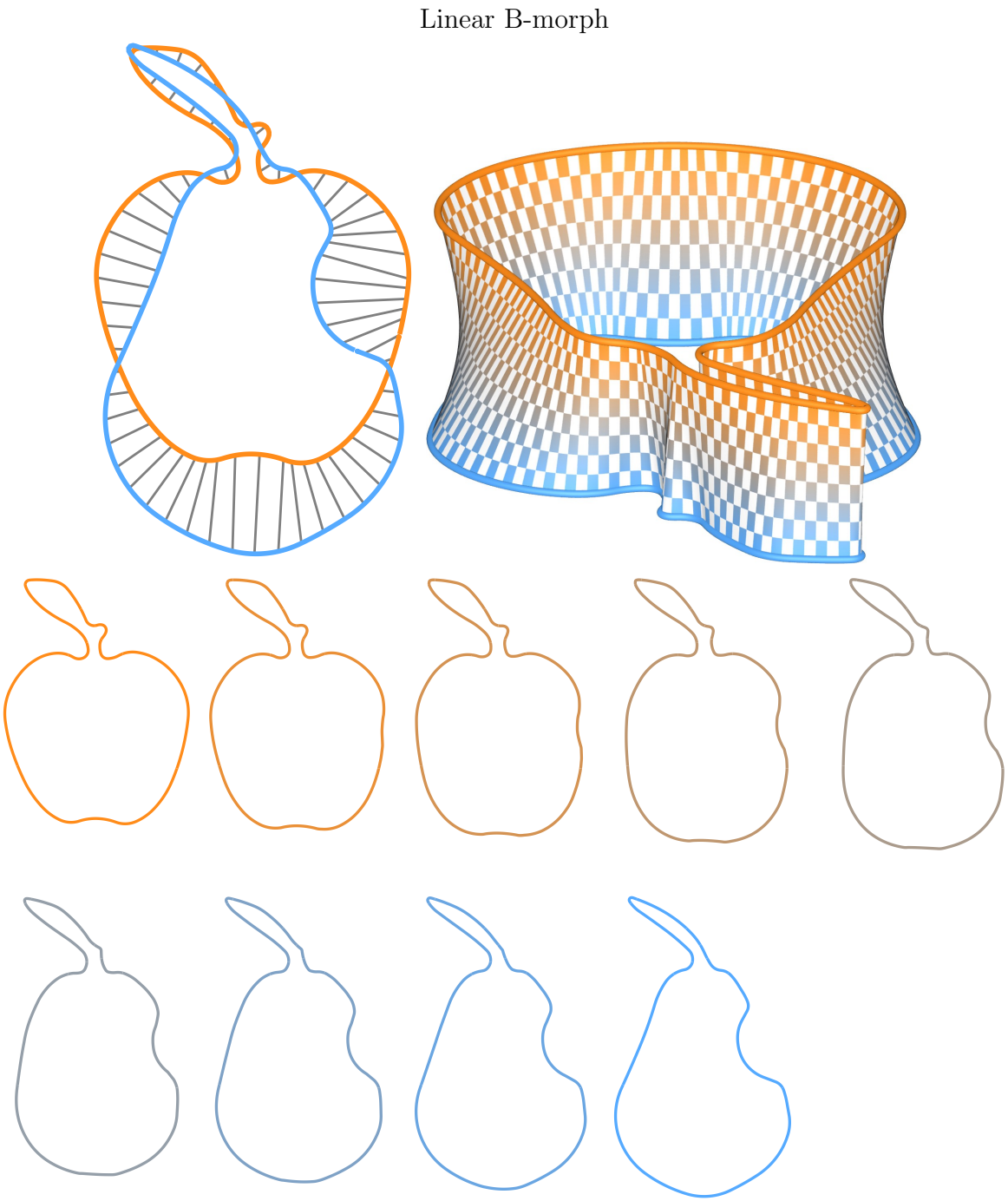


Figure 70: The result of using the Linear B-morph morphing algorithm

Circular B-morph

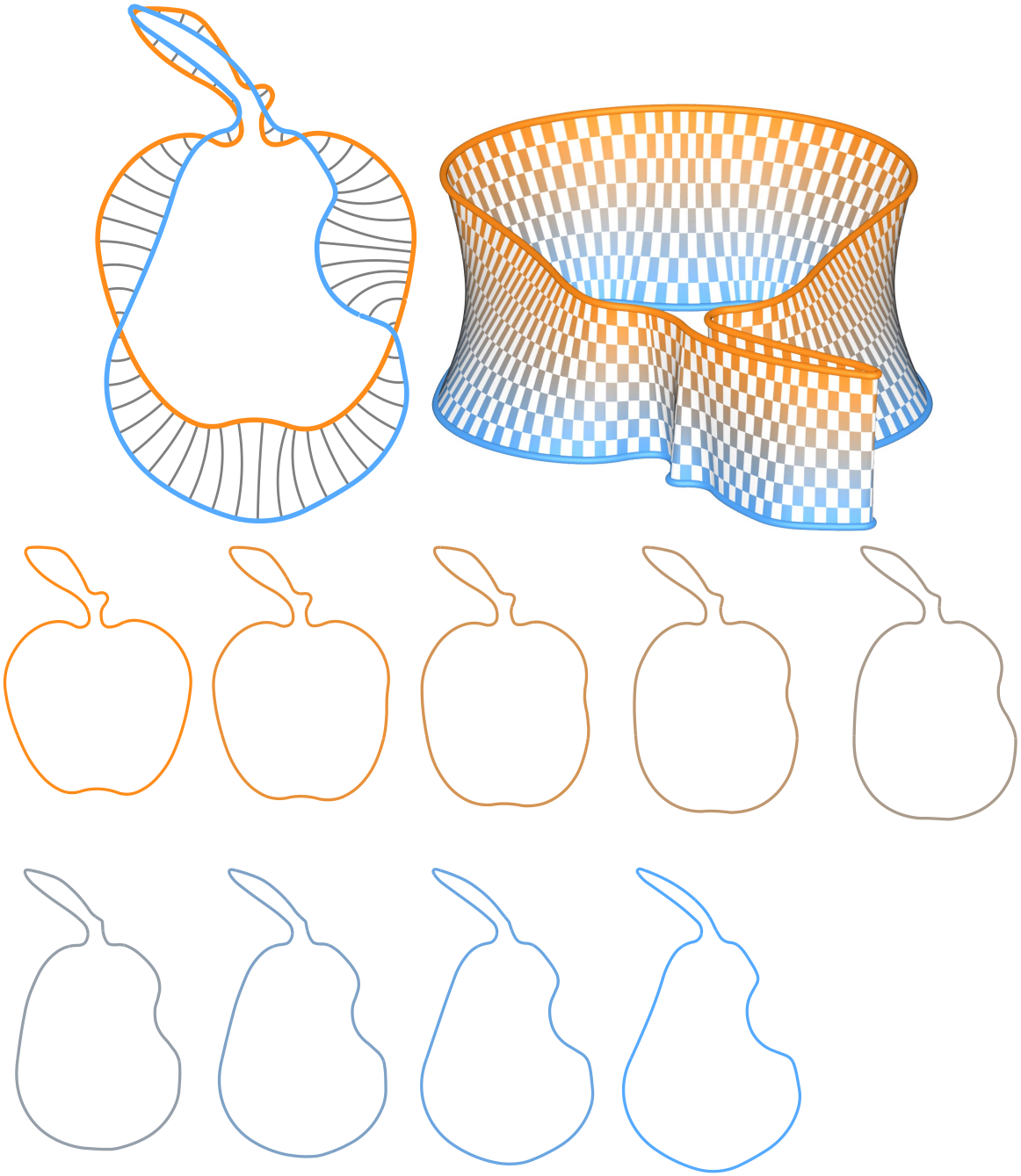


Figure 71: The result of using the Circular B-morph morphing algorithm

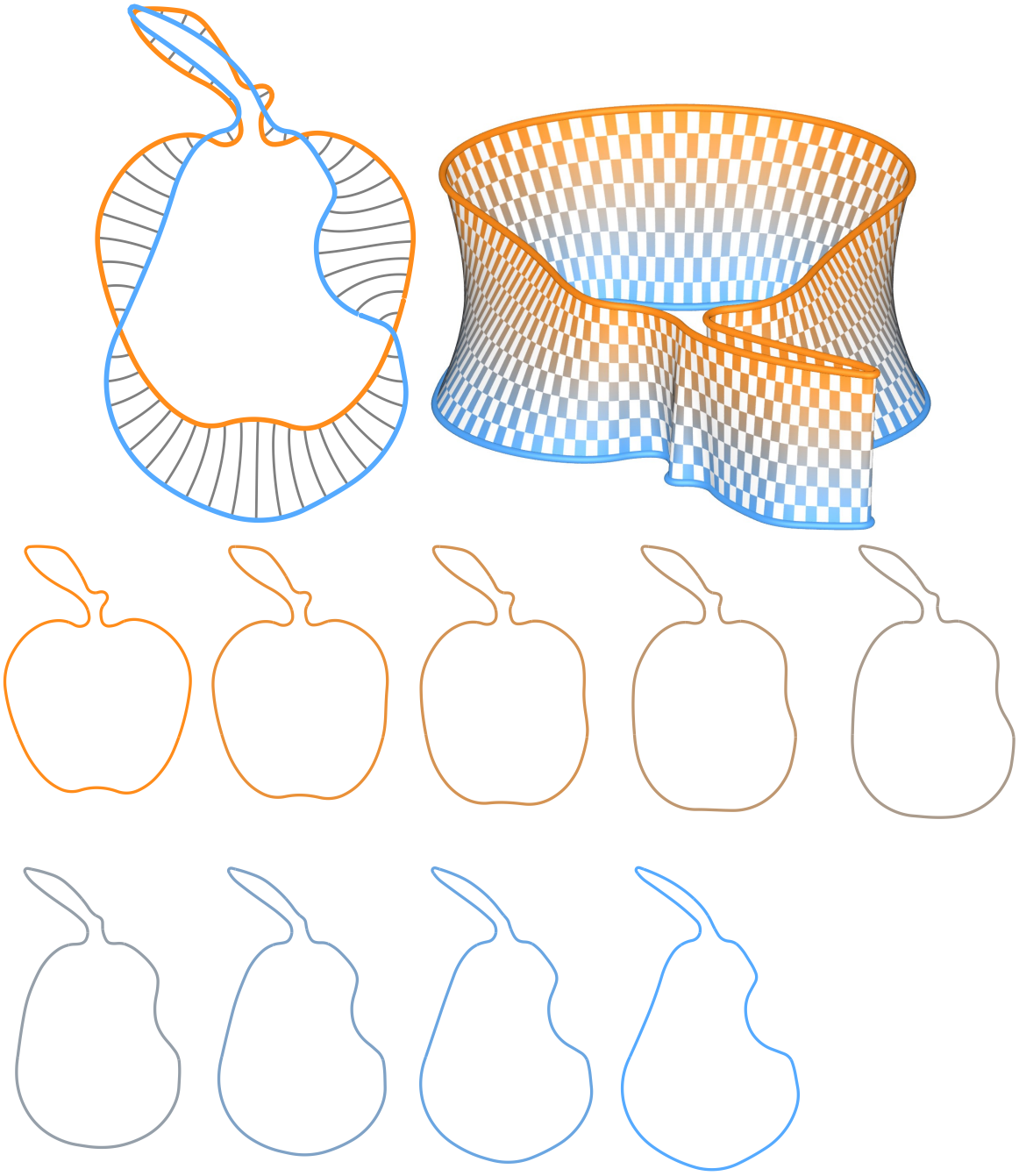


Figure 72: The result of using the Heat Propagation morphing algorithm

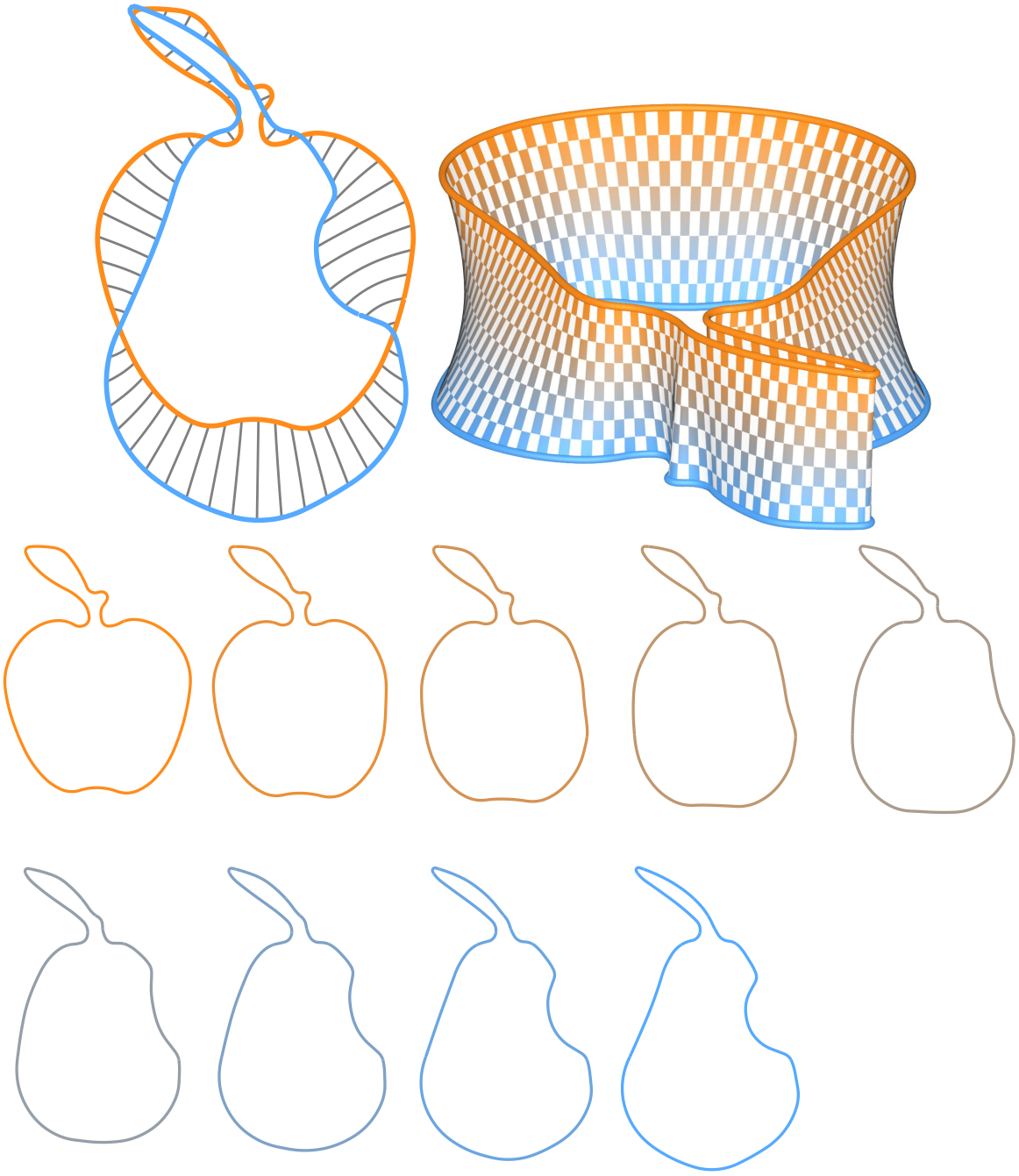


Figure 73: A result of using the Laplace Blending morphing algorithm on a pair of offset circles.

8.4 Conclusion

Stretch is a measure where none of the *b-morphs* fare well. This is due to its local, orientation-based correspondence. This effect is also exhibited by the *Heat Propagation* morph. As shown in Figure 8.3, an unnatural "bend" may occur in areas of

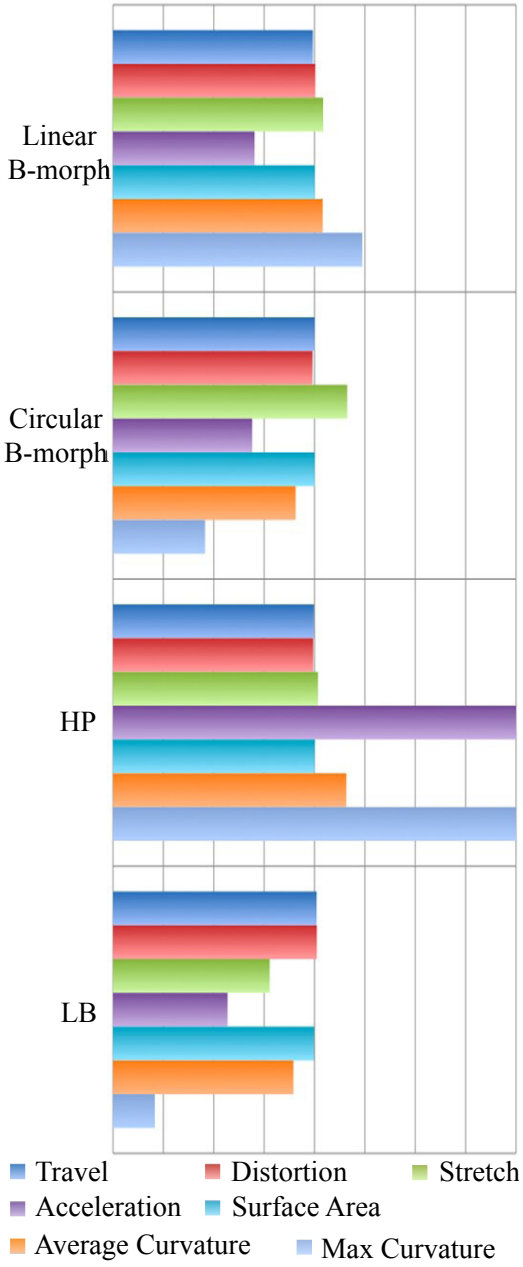


Figure 74: The measures for each of the 4 methods used in the Apple to Pear morph. Each measure is normalized independently by dividing by the maximum result.

high *stretch*. Although the resulting morph curves are smooth, the bend immediately draws the attention of the viewer. The *stretch* is lower for the other morphs because we use uniform sampling to define their correspondence. For example, in the 'S'-curve example, each edge segment starts and ends with the same length when uniform sampling is used.

The effect of *distortion* for the morphs is made evident when examining the checkerboard texture applied to the corresponding surface reconstruction. The *Circular b-morph* and *Heat Propagation* morph result in a texture where the gridlines remain almost exactly orthogonal. In the other morphs, the checkboard is more distorted, having angles which deviate farther from $\pi/2$.

Throughout the results, it is clear that the *Circular b-morph* and the *Heat Propagation* morph produce similar results in terms of appearance and measure. Both use trajectories that are orthogonal to the input curves. The *Heat Propagation* morph theoretically should have zero distortion since every point moves in the direction of steepest descent along the gradient defined by the heat equation within the *moat* X. However, due to the discretized nature of our implementation of this approach, this is not the case. Our *Circular b-morph* approximates this property of following the normal flow. We argue that for the case of *b-compatible* curves, the *Circular b-morph* offers a reliable alternative to the *Heat Propagation* morph for minimizing *distortion*. It is easier and faster to compute and can be computed exactly (using closed form expression) in the case of PCCs.

The *Curvature Interpolation* morph and *Laplace Blending* morph perform well in terms of average mean curvature of the resulting surfaces. Indeed, by design they interpolate the curvature defined by the input curves. The *Laplace Blending* approach does this best, as shown in Figure 8.3 where the shape remains circular throughout the morph. Our *Curvature Interpolation* morph produces ellipses for this example (Fig. 8.3) due to the final 1D scale operation we use to bring the endpoints into alignment. The *Circular b-morph* offers competitive results for mean curvature. As previously stated, for C^k continuous input curves, the inbetween *b-morph* curves are guaranteed to be at least C^{k-1} continuous for $k \geq 2$, and the trajectories in 3D are C^1 continuous circular helices.

Because the choice of morph is application dependent and inherently subjective,

we do not argue that any one measure is more important than another. However, for applications where the input curves are *b-compatible*, and *distortion*, *acceleration*, *average curvature*, and *travel distance* are important, the *b-morphs* offer important advantages over the others benchmarked here.

CHAPTER IX

BALL-MAP AND BALL-MORPH EXTENSIONS TO INCOMPATIBLE SHAPES

As discussed in previous chapters, the *ball-map*, and consequently the *b-morph*, is restricted to cases where the input curves are *b-compatible*. We discuss here an extension in order to accommodate incompatible features for curves. We assume that the curves A and B fall into one of the three same categories as defined in Chapter 3:

1. Open curves with shared endpoints that meet at angles less than π
2. Closed curves such that $iA \cap iB$ and $eA \cap eB$ are each a non-empty connected set
3. Closed curves such that $A \cap B = \emptyset$ and $A \cap iB = A$ or $B \cap iA = B$

And we again assume that $i!X(A, B)$ has no more than 2 components and the interior medial axis of X is simply connected.

9.1 Identifying incompatible features

Finding incompatible features is equivalent to finding branches in the medial axis of the *moat* of two shapes. These branches form at bifurcation points on the medial axis. These bifurcation points exist where a ball has tangential contacts with more than one point on one of the two shapes. If the input shapes are closed piecewise-circular curves, then Apollonius' circle solution can be used in a greedy search to find these points by trying every combination of three circular arcs. A valid result is one which has a tangential contact with each of the three circular arcs and intersects no other arcs of either curve. This algorithm is defined in detail in Chapter 3, Algorithm 1.

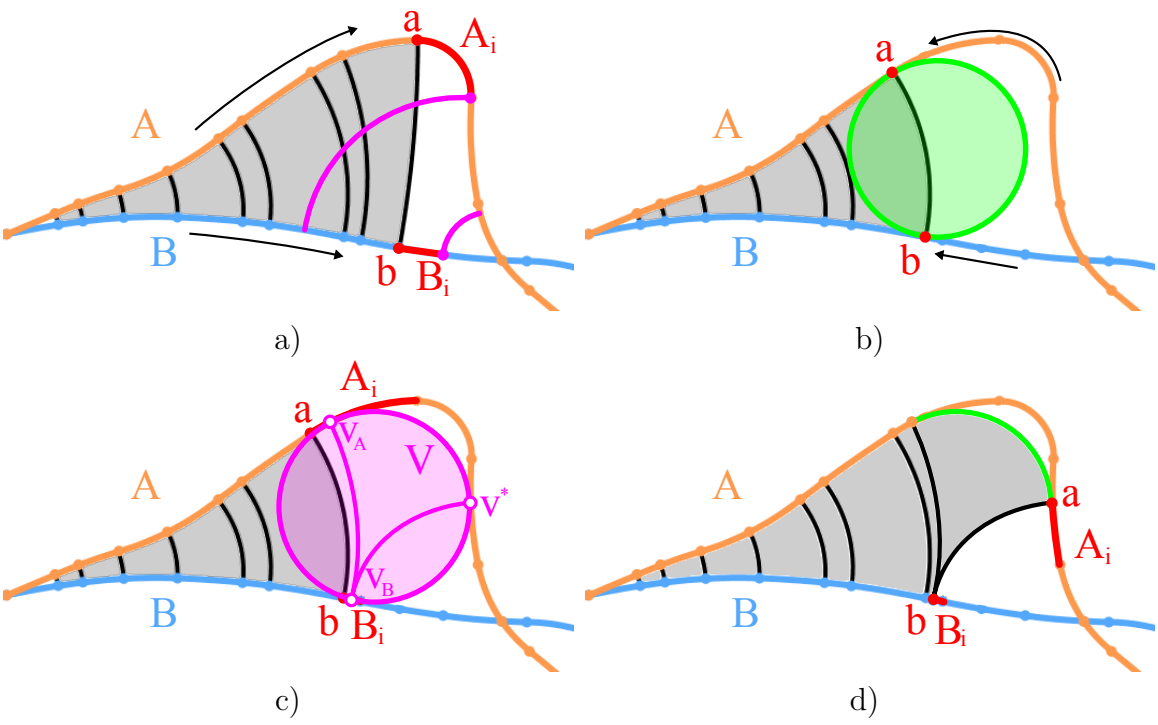


Figure 75: a) The lacing process has passed an incompatible feature since a *ball-map* solution does not exist between a and B_i or b and A_i. b) a and b step backward until a *ball-map* solution is found that does not intersect A or B. c) Using the Apollonius' circles solution, a *ball-map* V is found by using A_i, B_i and testing with every other arc on both curves. d) The *ball-map* process continues by moving a and b past the incompatible feature.

We propose a modification of the *ball-map* lacing algorithm in Chapter 5 such that it will detect incompatible features and construct the relative blending arcs (Chapter 4), then continue the lacing process.

Assume that A and B are manifold curves and that $iA \cap iB$ and $eA \cap eB$ are each a non-empty connected set. Once the lacing process reaches points $\mathbf{a}, \mathbf{b}$ and associated arc-edges A_i, B_i such that no *ball-map* solutions exist which map $\mathbf{a}$ to B_i or $\mathbf{b}$ to A_i, an incompatibility has been found (Figure 75-a). Instead of returning an error as before, the $\mathbf{a}$ and $\mathbf{b}$ pointers step backward simultaneously to previously computed *ball-map* solutions until one is found such that its tangent ball does not intersect A or B. At each backward step, the ball corresponding to the previously computed map from $\mathbf{a}$ to $\mathbf{b}$ is checked for intersection with A and B ($\mathrm{O}(n)$). The

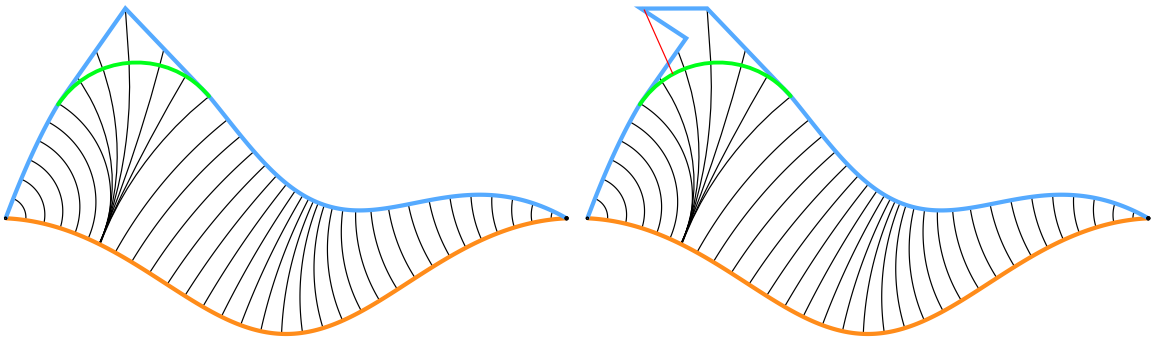


Figure 76: Left: *ball-map* arcs between the *relative blended* versions of the curves and *Closest Projection* linear trajectories from the blue curve to its relative blended version. Right: The *Closest Projection* morph is not a homemorphism, so additional *relative blending* operations will be necessary.

process continues until a *ball-map* is found that does not intersect A or B ($\mathrm{O}(n^2)$ in the worst case) as shown in Figure 75-b. At this point, a tangent ball $\mathbb{V}$ must exist that is tangent to the arc-edges A_i and B_i corresponding to a and b. $\mathbb{V}$ will also be tangent to additional point(s) on A and/or B. The Apollonius' circles solution is constructed using the arc-edges A_i, B_i and every other arc-edge on A and B that has not already been laced until the solution tangent ball $\mathbb{V}$ is found that does not intersect A or B ($\mathrm{O}(n^2)$ in the worst case) and is tangent to A at $\mathbf{v}_A$, B at $\mathbf{v}_B$, and either A or B at $\mathbf{v}^*$ as shown in Figure 75-c. If $\mathbf{v}^*$ lies on A, then the shortest circular arc on $\mathbb{V}$ between $\mathbf{v}_A$ and $\mathbf{v}^*$ is the relative blending arc for the incompatible feature. Finally, $\mathbf{a}, \mathbf{b}$ are set to $\mathbf{v}^*, \mathbf{v}_B$, respectively, and the lacing process continues, as shown in Figure 75-d.

Once all vertices on A and B are laced, the relative blended curves $R_B(A), R_A(B)$ have been computed.

9.2 Composite b-morphs in 2D

To produce morphs between curves that are not *b-compatible*, we compute the *relative blendings* $A' = R_B(A)$ and $B' = R_A(B)$ and the *b-morph* M_0 between the resulting *quasi-b-compatible* curves A' and B' using the lacing algorithm from Chapter 5.

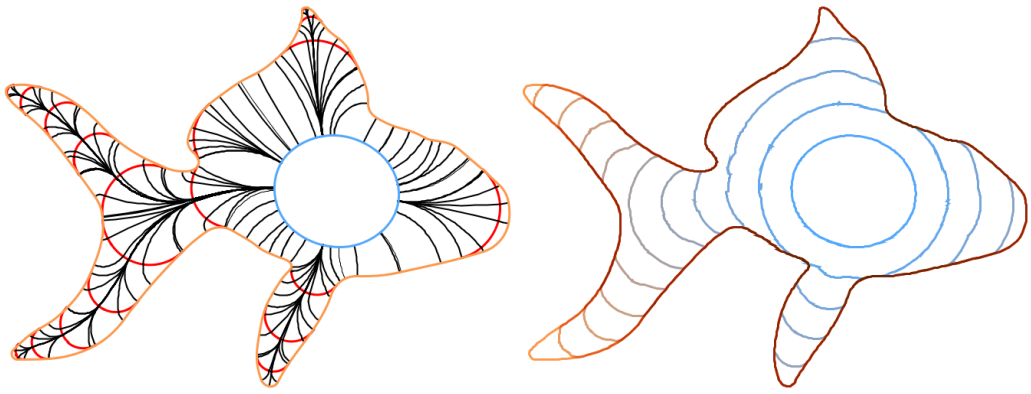


Figure 77: Trajectories (left) shown with caps (red) computed by recursive *relative blending* operations and constant speed morph curves from blue to orange (right).

There are 3 situations for computing the next morph M1:

1. If B' and B are *b-compatible* we compute their *b-morph* M_1.

2. If B is not at least C^1 continuous, we try a *Closest Projection* morph (discussed in Chapter 8) from B to B'. If the closest projection is a homeomorphism, we produce a morph M_1 with the reversed straight line trajectories (Figure 76-left)

3. If B is at least C^1 continuous, or the the closest point projection is not a homeomorphism (Figure 76-right), we compute the *relative blending* $B'' = R_{B'}(B)$, then compute their *b-morph* M_1.

Note that in all cases, the trajectories of M_1 leave B' along its local normals and are hence smoothly joined with the trajectories of M_0. If we run into situation (3), we recurse on the gap between B'' and B. This process fills the gap between B' and B by a series of *b-morphs* and possibly a final *Closest Projection* morph, generating piecewise-circular trajectories (Fig. 77 left) with possibly a straight line at the end of each one. Note that without handling situation (2), the recursive process would never converge since a ball will never reach the sharp feature. We have produced a concatenation of morphs M_1, M_2, M_3... We perform a similar iterative process to invade the gap between A' and A, producing in this manner a series of morphs, which

we reference with negative integers: M_{-1}, M_{-2}, M_{-3} The final combined morph is: M_{-3}, M_{-2}, M_{-1}, M_0, M_1, M_2, M_3 (Fig. 77 (right), Fig. 9.2, Fig. 9.2, Fig. 9.2).

We have explored several synchronizations of the motions of sample points of A along their smooth trajectories towards B during a composite morph (Figs. 9.2, 9.2, 9.2). The first one performs one morph after the other in the sequence M_2, M_1, M_0, M_1, M_2 as shown in Figure 9.2. This has the effect of shrinking and growing the incompatible features first and last. Also notice that there is a large jump between frames 5 and 6. Each "stage" of the morph is interpolated with the same timing and because the largest change occurs when morphing the M_0 stage between the relative blendings of the original curves, it moves very quickly.

The second synchronization moves each sample point of A at uniform speed along its composite trajectory (Fig. 9.2). Notice the incompatible features shrink quickly to the medial axis, producing non-smooth results.

The third synchronization moves all samples with the same speed, synchronized to reach the N curve (Chapter 7, Figure 45) between A' and B' at the same time (Fig. 9.2 right). This is a compromise between the other two methods and produces the best results, although non-smooth morph curves are still be produced.

9.3 Limitations

As shown in the various synchronizations of the composite *b-morph*, the inbetween curves are not smooth. The composite *ball-map* produces trajectories that do not self-intersect, although choosing a parameterization of these trajectories that ensures C^1 continuity has been left for future work. We hope that the fast and precise computation of the trajectories will be useful in other applications, such as measuring tissue thickness in medical imaging analysis [133].

As is the case for the compatible *b-morphs*, the composite morph curves exist entirely within the bounds of the *moat*, and the points of intersection of the input

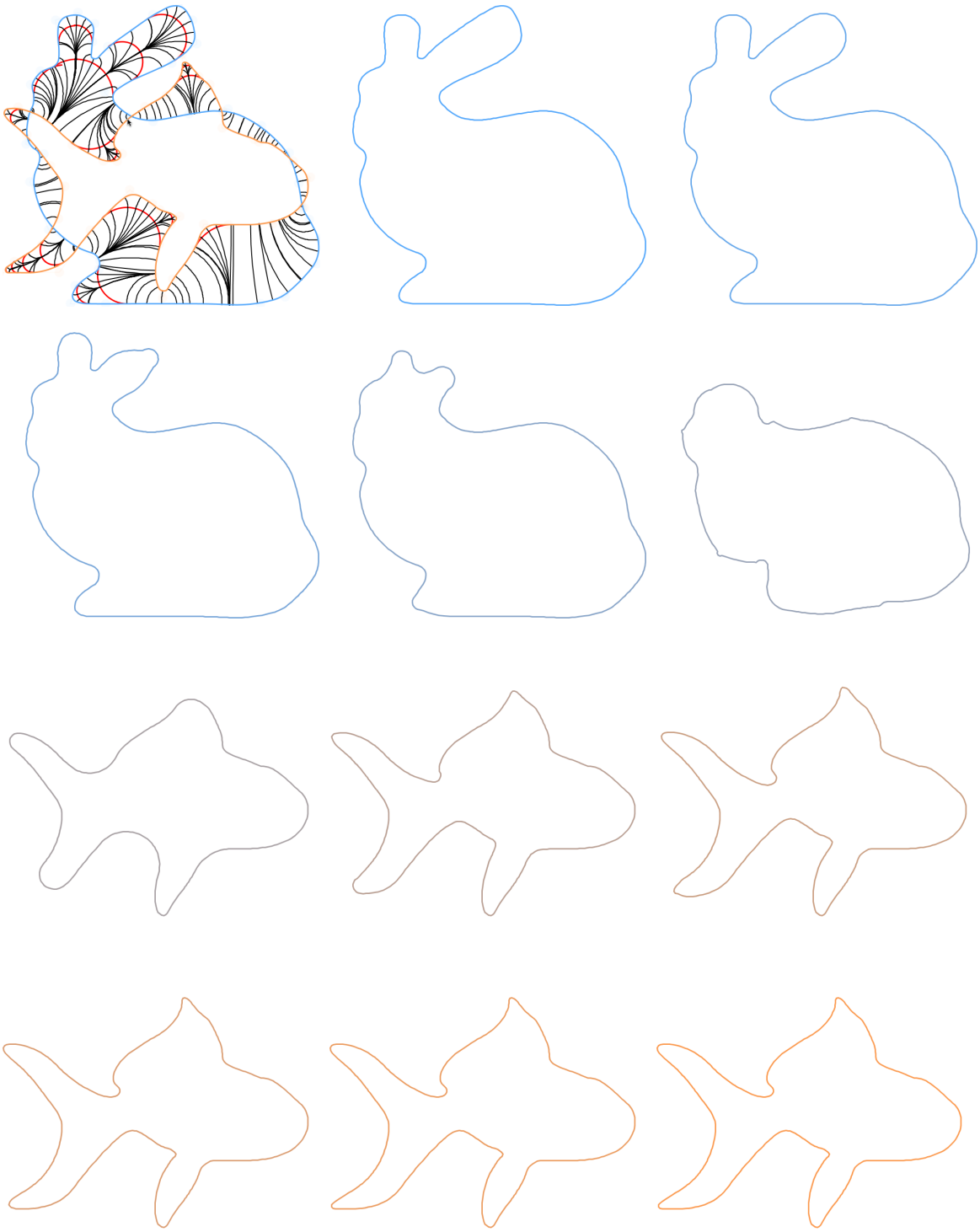


Figure 78: Composite *b-morph* and trajectories (black, top-left) between incompatible shapes obtained by recursively applying the *relative blending* operation and morphing each stage of the trajectories one level of a recursion at a time (right).

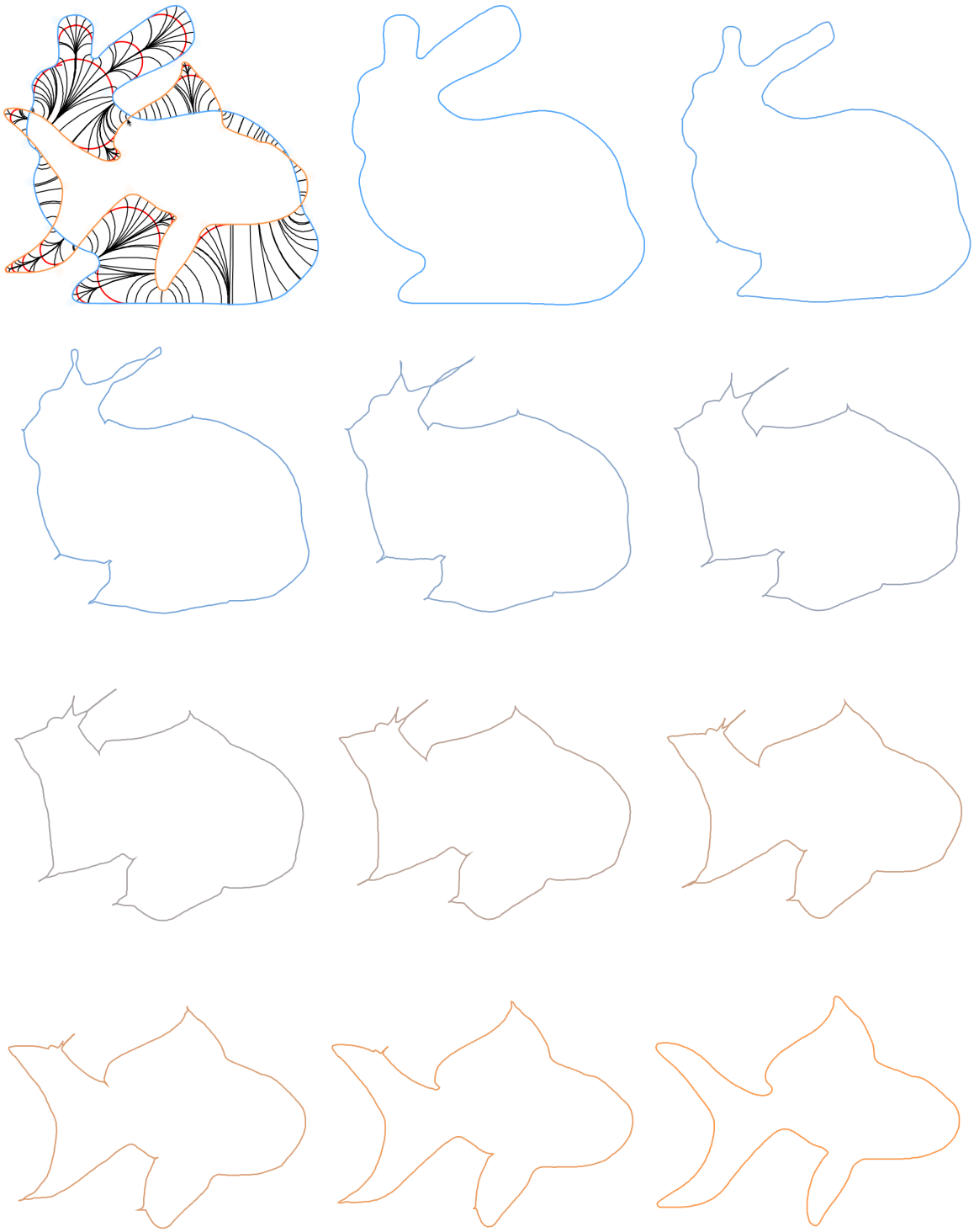


Figure 79: Composite *b-morph* and trajectories (black, top-left) between incompatible shapes obtained by recursively applying the *relative blending* operation and morphing along the uniformly sampled trajectories (right).

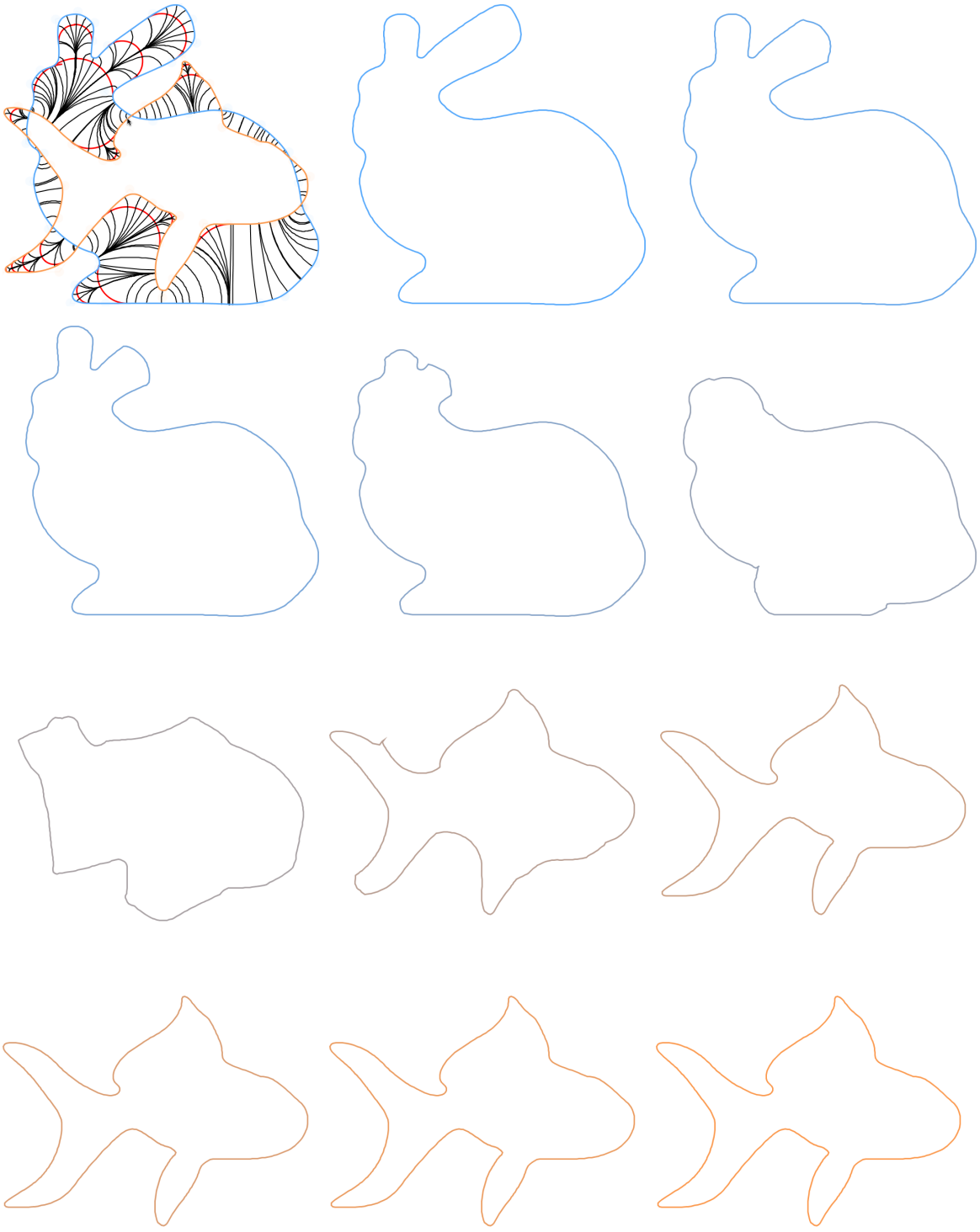


Figure 80: Composite *b-morph* and trajectories (black, top-left) between incompatible shapes obtained by recursively applying the *relative blending* operation and moving all samples at uniform speed such that they are synchronized to reach the N curve between A' and B' at the same time (right).

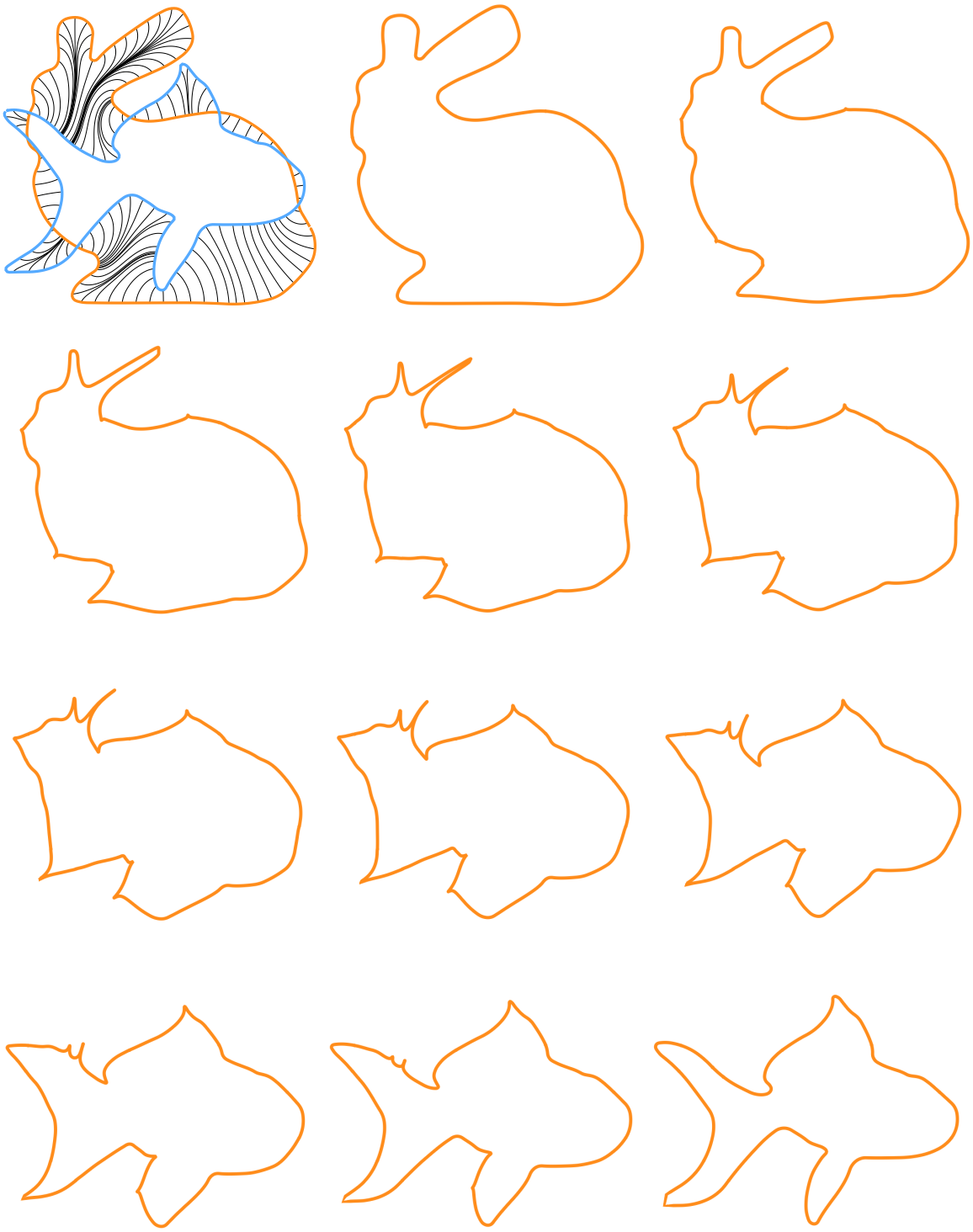


Figure 81: The *Heat Propagation* morph produces similar results to the uniform speed composite *b-morph* (Figure 9.2). Notice that it also produces thin features in the inbetween curves.

curves remain fixed throughout the morph. This effect may not be desirable in some applications, and another morph, such as one discussed in Chapter 8, may be used.

9.4 Analysis

As discussed in Chapter 8, the *Circular b-morph* produces similar results to the *Heat Propagation* morph. For the case of incompatible curves, the *Heat Propagation* morph (Figure 9.2) produces results similar to the uniform speed composite *b-morph* (Figure 9.2). Again, each has trajectories that are orthogonal to the input curves. Also, the trajectories exist entirely in the *moat* X. When parameterizing the trajectories uniformly, both morphs result in features shrinking to curves with sharp C^0 continuous features. However, the *Heat Propagation* morph does produce a one-to-one correspondence whereas the *ball-map* is one-to-many where the curves are incompatible.

The advantage of the *b-morph* in these incompatible cases is precision. When using PCCs, we have a closed-form solution for computing the correspondence and trajectories. As previously mentioned, the resulting trajectories will also be piecewise-circular curves.

CHAPTER X

PEARLING

Pearling is a novel segmentation technique for the semi-automatic extraction of idealized models of networks of strokes (variable width curves) in 2D images and variable radius networks of tubular structures in 3D medical images. *Pearling* is related to the *ball-map* construction in that we want to compute an approximate medial axis transform, but instead of having explicit geometry, we have discrete models (pixels, voxels). The approximate MAT may for example represent roads in an aerial photograph, vessels in a medical scan (image or volume), or an artist's strokes in a scanned drawing of feature animation. For clarity, we use "vessel" to describe any traceable path (i.e. roads, arteries, strokes).

10.1 Overview

The operator seeds the *pearling* process by selecting representative areas of good (vessel to be traced) and bad colors (or gray levels). Then, the operator may either provide a rough trace through a particular path in the vessel graph or simply pick a starting point (seed) and a direction of growth. *Pearling* computes in realtime the centerlines of the structures, the bifurcations, and the thickness function along each structure, hence producing a purified approximating medial axis transform of a desired portion of the graph. No prior segmentation or thresholding is required. Simple stylus gestures may be used to trim or extend the selection or to add branches. The realtime performance and reliability of *pearling* results from a novel ball-sampling approach, which traces the structures by optimizing the positions and radii of a discrete series of balls (pearls) along the structure. By design, the idealized pearl string model is slightly wider than necessary to ensure that it contains the boundary. A narrower

core model that fits inside the structure is computed simultaneously. The difference between the pearl string and its core, which we call the *crust*, contains the boundary of the structure and may be used to capture, compress, visualize, or analyze the raw image data along the boundary. For example, the core may trace the blood in an artery, while the crust will capture the arterial wall.

10.2 Prior Art

Given its clinical importance, the problem of vessel segmentation has received a fair amount of attention in the literature. Kirbas et al. [63] provide a recent survey of techniques for vessel segmentation. They conclude that there is no single vessel segmentation approach that is robust, automatic, and fast, and that successfully extracts the vasculature across all imaging modalities and different anatomic regions. Following their conclusion, we step away from total automation and instead focus on interactive segmentation, providing an efficient system for an operator to quickly extract the region and branches of interest.

Other recent vessel segmentation approaches in the medical imaging literature include [116], which models vessel segments using superellipsoids, [73], which utilizes co-dimension two level set flows, and [109], which applies a Bayesian classifier to feature vectors produced using Gabor wavelets. While elegant, these techniques require significant computational resources.

Fast methods such as [128] perform the segmentation on slices made in a direction orthogonal to the vessel centerline or intensity maxima [111] and then extract the vessel geometry by connecting the results. More closely related to our work is [77], which builds tunnels modeled as a union of spheres, placed through protein molecules along the edges of a Voronoi diagram computed from the atoms (represented by spheres). Unlike this work, our method is image-based and designed for segmentation, placing the pearls using local pixel intensity inside each pearl.

Several authors proposed to compute skeletons through the advection of particles along the distance field [35] [113]. The thinning methods [90] and the methods based on distance transform [117] for extracting a skeleton are not easily applicable to our problem of rapidly segmenting a subset of the volume because they are global and hence slow and require the precomputation of a distance field. The distance field precomputation may be avoided in methods that use a general scalar field [34]. Minimal paths through vessels have been computed using fast marching techniques [69] that backtrack along a distance field [31] computed with a Riemannian metric.

Also related is the computation of the medial axis transform [17], as discussed in previous chapters. Once an image is segmented and pixels have been identified that are in the vessel network, other techniques for producing the medial axis transform on discrete data could be used. However, *pearling* produces directly the purified MAT that could be derived by removing automatically small branches from the MAT and selecting a desired portion of the graph, all without requiring a prior segmentation.

10.3 Our contribution

Pearling performs a segmentation and idealization simultaneously by computing an ordered series of *pearls*, which are balls of possibly different radii. Starting with an initial pearl given by the operator, as well as an initial direction, *pearling* iteratively computes the position and radius of an adjacent pearl based on the image data, so that the newly placed pearl fits properly along the desired path, slightly bulging out of the segmented shape on both sides. The method proceeds in this fashion, producing a string or network of strings of pearls that provide a discrete representation of the vessel geometry. A final smooth contour representing the vessel network is then obtained by estimating continuous functions that interpolate the discrete series of pearls. As we will show, *pearling* is computationally efficient and well suited to user interactivity. This interactivity affords operator guidance of the segmentation in a

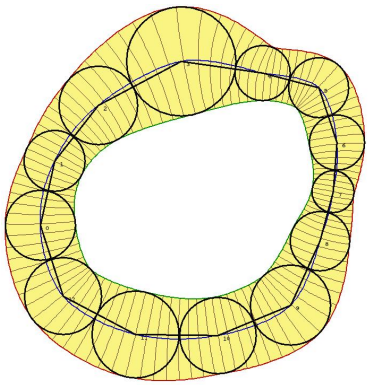


Figure 82: Representation of *pearling*, which consists of an ordered series of pearls. Continuous functions that interpolate the pearls are also shown: the blue curve interpolates the pearl centers, and the red and green curves interpolate the outside and inside edges of the pearls, respectively. Note the circular arc trajectories of the *b-morph* joining corresponding points on each side.

particular direction as well as operator correction of errant segmentation results.

In addition to pearls, we introduce the concepts of core and crust. The core of the *pearling* model is computed by reducing the radius of each pearl while keeping its center in place. The radius is chosen to be as large as possible, while ensuring that a given majority of pixels inside the core pearl are good. The difference between the "outer" pearl model and its core is the crust, which may be used to capture, compress, archive, transmit, visualize, or analyze the original data along the vessel border. We show in Section 10.5 that the crust is a narrow band around the vessel border computed through region growing techniques.

10.4 Methodology

Pearling allows the operator to extract a higher level idealized parametric representation of each vessel. This representation is called a *string*. As shown in Figure 82, it comprises an ordered series of pearls (or balls), each defined by the location $\mathbf{c}_i$ of its center, by its radius r_i, and a time value t_i, and possibly other attributes a_i. The continuous model of the corresponding vessel that is recovered by *pearling* is defined

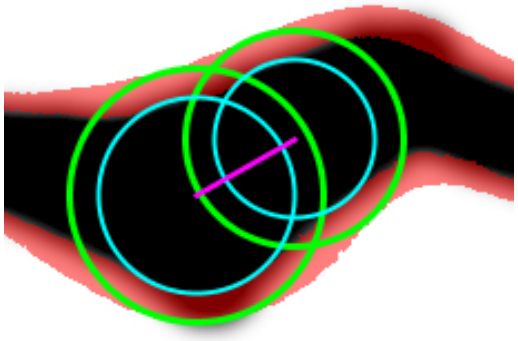


Figure 83: Two adjacent Pearls (green) and their cores (cyan) with their centers connected (magenta) overlaid on top a vessel (black) and the crust region defined by the continuous set of pearls minus their cores (red)

as the region W swept by a pearl whose center $\mathbf{c}(t)$ and radius $r(t)$ are both continuous and smooth functions of the time parameter t. These functions interpolate the centers and radii of the string of pearls for given values t_i of time and define the MAT for the string. For a network of vessels, multiple strings will be defined which meet at bifurcation pearls, the centers of which define bifurcation points of the MAT. Bifurcation pearls will be defined later.

10.4.1 Estimation of pearls

As previously discussed, *pearling* starts with an initial seed, $\mathbf{c}_0$, which is a point either provided by an operator or by an algorithm. Typically, $\mathbf{c}_0$ should be chosen to lie close to the centerline of a path in the image and possibly at the end of one of the paths of the desired structure. *Pearling* then uses an iterative process to construct an ordered series of pearls, one at a time. During that process, at each step, the center $\mathbf{c}_i$ and radius r_i of the current pearl is chosen so as to maximize r_i subject to image data, given the constraint that the distance d_i between $\mathbf{c}_i$ and the center $\mathbf{c}_{i-1}$ of the previous pearl is bound by functions $d_{min}(r_{i-1}, r_i)$ and $d_{max}(r_{i-1}, r_i)$. We use linear functions $d_{min,max}(r_{i-1}, r_i) = ar_{i-1} + br_i$, and with bounds $d_{min}(r_{i-1}, r_i) = r_i/2$ and $d_{max}(r_{i-1}, r_i) = r_{i-1} + r_i$. It is important that d_i be allowed to fluctuate in order to

capture rapid changes in thickness and allow convergence to local minima.

Let $\mathbf{c}_{i-1}$ be the center of the previous pearl, as shown in Figure 83. Let $\mathbf{c}_i$ be the center of the next pearl, and let r_i be its radius. The objective is to find the optimal values of $\mathbf{c}_i$ and r_i that place the ith pearl in the vessel. In order to adjust the values of r_i and c_i so that the pearl fits more "snugly" astride the vessel, we define two functions: $\mathbf{f}(\mathbf{c}_i, r_i)$ and $g(\mathbf{c}_i, r_i)$, as described below.

Center estimation: The function $\mathbf{f}(\mathbf{c}_i, r_i)$ returns a gradient vector indicating the direction in which $\mathbf{c}_i$ should be adjusted and the amount of the adjustment. $\mathbf{f}(\mathbf{c}_i, r_i)$ takes the form:

$$\mathbf{f}(\mathbf{c}_i, r_i) = \frac{30}{7\pi r_i^2} \int_{\mathbf{x}\in P_i} \phi(\mathbf{x})(\mathbf{c}_i - \mathbf{x}) \left(1 - \frac{\|\mathbf{c}_i - \mathbf{x}\|^2}{r_i^2}\right) d\mathbf{x} \tag{28}$$

for *pearling* in 2D, and

$$\mathbf{f}(\mathbf{c}_i, r_i) = \frac{5}{\pi r_i^3} \int_{\mathbf{x}\in P_i} \phi(\mathbf{x})(\mathbf{c}_i - \mathbf{x}) \left(1 - \frac{\|\mathbf{c}_i - \mathbf{x}\|^2}{r_i^2}\right) d\mathbf{x} \tag{29}$$

for *pearling* in 3D.

$$\phi(\mathbf{x}) = \begin{cases} 1, & \text{if } \hat{p}_{\text{out}}(I(\mathbf{x})) > \hat{p}_{\text{in}}(I(\mathbf{x})) \\ 0, & \text{otherwise} \end{cases} \tag{30}$$

The function $\mathbf{f}(\mathbf{c}_i, r_i)$ sums the vectors $(\mathbf{c}_i - \mathbf{x})$, where $\mathbf{x}$ is the vector coordinate of the the current grid cell (pixel or voxel), across the entire area of cells for pearl P_i, using only those cells such that $\phi(\mathbf{x}) = 1$, i.e., cells determined to be outside the vessel. Each of these vectors is weighted by its distance from $\mathbf{c}_i$ such that cells nearer $\mathbf{c}_i$ have a stronger influence on the result, as reflected in the $\left(1 - \frac{\|\mathbf{c}_i - \mathbf{x}\|^2}{r_i^2}\right)$ component of Equations 28 and 29. Intuitively, Equation 30 states that each point inside the ith pearl but outside the vessel imparts a force on the pearl that pushes it away from the vessel boundary. When the forces are balanced on all sides of the pearl, the pearl is typically centered in the vessel.

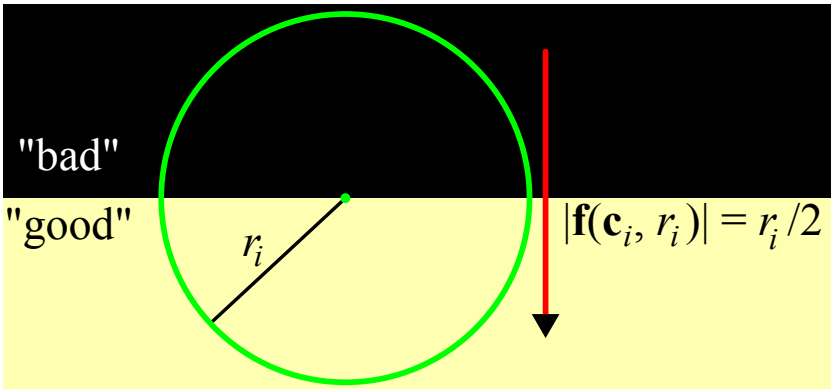


Figure 84: The normalization factors $\frac{30}{7\pi r_i^2}$ and $\frac{5}{\pi r_i^3}$ are such that in this ideal example, $\mathbf{f}(\mathbf{c}_i, r_i)$ has magnitude $r_i/2$ and offsets the pearl half the distance into the "good" region.

The $\frac{30}{7\pi r_i^2}$ and $\frac{5}{\pi r_i^3}$ are normalization factors are computed by examining the case where a pearl is cut into two equal halves by a linear boundary separating good and bad cells, as shown in Figure 84. By assuming this ideal case, we can compute the normalization factor by calculating the offset magnitude needed to move this pearl half the distance that would move it entirely in the good region, which is $r_i/2$.

Determination of whether a cell lies inside the vessel or outside the vessel is necessary when computing $\mathbf{f}(\mathbf{c}_i, r_i)$, and is achieved using non-parametric density estimation. Before running the algorithm, the operator selects two regions; one inside the vessel and one outside. For a given region, we estimate the density by applying a smoothing kernel K to the cells in the region's histogram, i.e.,

$$\hat{p} = \frac{1}{n} \sum_{i=1}^{n} K\left(\frac{I_i - m}{h}\right), \tag{31}$$

where I_i is the intensity of the ith cell in the region, m is the mean of intensities of the n cells in the region, and h is the bandwidth of the estimator. We use a Gaussian kernel, $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$. Performing this estimation on two operator-supplied regions results in two densities, $\hat{p}_{\text{in}}(I)$ and $\hat{p}_{\text{out}}(I)$. During segmentation, an intensity I is classified as outside if $\hat{p}_{\text{out}}(I) > \hat{p}_{\text{in}}(I)$; otherwise it is classified as inside.

Radius estimation: r_i is adjusted to better fit the vessel using the function $g(\mathbf{c}_i, r_i)$, as shown in Equation 32. For robustness, pearls are designed to have a percentage p of their cells inside the vessel and the rest outside the vessel, as indicated in Figure 83. In our implementation, $g(\mathbf{c}_i, r_i)$ is positive when less than p percent of the pearl's cells fit in the vessel and negative more than $1 - p$ of the cells lie outside of the vessel. The result of $g(\mathbf{c}_i, r_i)$ is then used to scale r_i to better fit in the vessel. $g(\mathbf{c}_i, r_i)$ takes the form,

$$g(\mathbf{c}_i, r_i) = p - \frac{\int_{\mathbf{x} \in P_i} (1 - \phi(\mathbf{x}))\, d\mathbf{x}}{\int_{\mathbf{x} \in P_i} d\mathbf{x}} \tag{32}$$

Interleaving the estimation, and convergence: We interleave the estimation of c_i and r_i for the ith pearl P_i. For a given position c_i and radius r_i, one can update both parameters independently using the results given by $\mathbf{f}(r_i, \mathbf{c}_i)$ and $g(r_i, \mathbf{c}_i)$. In both cases, the quality of the fit is measured and the desired adjustment is computed and returned.

The adjustments of c_i and r_i are done through several iterations while enforcing the constraint on d_i until the adjustment values returned by $\mathbf{f}(\mathbf{c}_i, r_i)$ and $g(\mathbf{c}_i, r_i)$ fall below a given threshold. The process then freezes the current pearl and starts fitting the next one. The growth of a vessel stops when the radius of the next pearl falls outside of a prescribed range, or when another application-dependent criterion is met, such as the detection of an operator-supplied endpoint or exceeding a prescribed arc-length of the spine. The result of this *pearling* process is a series of location-radius pairs $(\mathbf{c}_i, r_i)$, which we call the control pearls and which approximates a portion of the MAT of the vessel.

10.4.2 Direction estimation and Bifurcations

Often in the applications targeted, images contain sharp turns and bifurcations where a path will branch into multiple paths and may even contain loops. We have extended

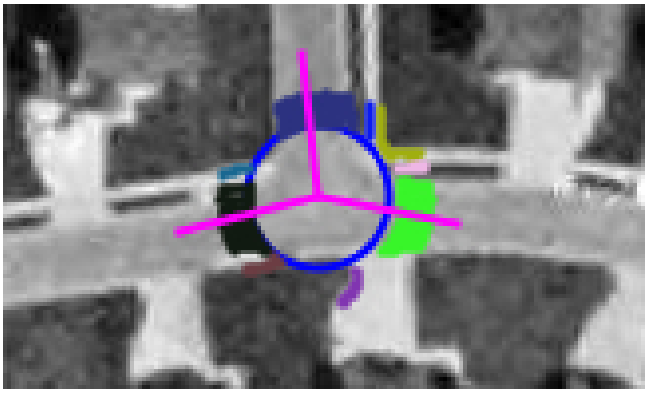


Figure 85: A pearl shown at a vessel bifurcation along with its set of good cells in the region bounded by r_i and kr_i. Each connected component C_j is uniquely colored. Vectors D_j (magenta) are shown for the connected components which contain a number of pixels greater than the set threshold.

the above process to support these options to trace a network of pearl strings. These networks are stored as directed graphs (in the direction of growth), which may also contain undirected closed loops in the case where the growing string intersects another string.

Once a pearl P_i has converged to fit on a vessel, an analysis of the region around P_i is done in order to decide where to initially position P_{i+1}, taking into account that there could be more than one P_{i+1} or none. This analysis is done on the pixels in a circular band around P_i in 2D, or a spherical band around P_i in 3D, with an outer radius kr_i where k is some constant. In our experiments, a value of $k = 1.35$ produces acceptable results. Each cell x in this band is then classified using Equation 30. The result gives a classification with $N > 0$ connected components of good pixels, as shown in Figure 85. At minimum, there should be a connected component identifying the path from the previous pearl, P_{i-1}. If $N = 1$, then the only possible direction is backwards and *pearling* stops growing the current branch because its end has been found. If $N > 1$, then the starting position(s) for any subsequent pearl(s) can be chosen using the remaining connected components. For each connected component C_j, an average point is calculated which defines a direction vector D_j from the pearl

center c_i. Subsequent pearls are then initialized to the points on the boundary points of P_i which intersect each D_j. This extension allows *pearling* to accommodate sharp turns and bifurcations in the vessel structure.

A loop in the *pearling* network can also exist. A simple intersection check between the current pearl and all previous pearls will reveal if such a loop has occurred. In this case, the current pearl is linked to the intersected pearl and its string growing process is ended. The intersected pearl could then become a bifurcation pearl, if it already has more than one adjacent pearl, or it could be the end of another stroke still in in the growing phase. If the latter is the case, and two growing strokes collide, they are both ended and become part of the same stroke.

10.5 Results

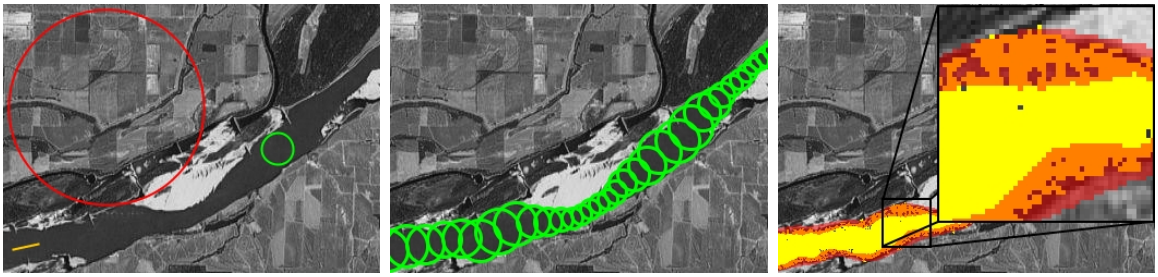


Figure 86: An unedited grayscale image obtained from satellite imagery of a river shown with the initial inputs(left), the resulting *pearling* segmentation, computed in 33ms(center), and the crust (red) overlaid atop a simple region-grown segmentation (yellow), showing that the majority of boundary details are contained within the crust.

We now present segmentation results using *pearling*. We begin with the segmentation of a satellite image of a river. The original image with the initial inputs is shown in Figure 86 (left). As described, the initial inputs include regions of good and bad pixels (shown in green and red) and an initial point and direction (shown in orange). The algorithm then proceeds, successively adding pearls until no more can be added, the result of which is shown in Figure 86 (center). From the collection

of discrete pearls and their cores, we can then extract the continuous model of the crust, as seen red in Figure 86 (right). The yellow region in the figure is the result of a simple region-growing segmentation using the same good and bad regions and starting point as *pearling*. A zoomed-in section shows almost all boundary pixels are contained within the *pearling* crust.

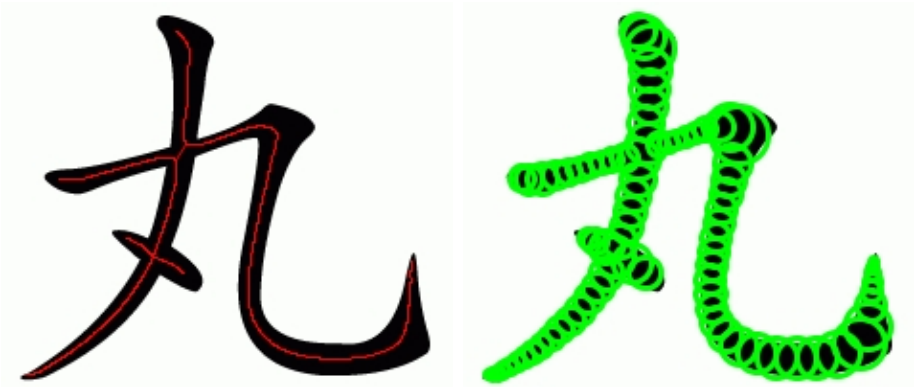


Figure 87: A Chinese character with centerlines (left) and discrete pearls (right) computed by *pearling* in 37ms on an image with size 236x201.

Figure 87 presents another segmentation result, of a variable width Chinese character. We show the original image with the *pearling*-computed centerlines (left) and the pearl disks (right). Note that all examples completed in tens of milliseconds.

Additional results for both 2D and 3D *pearling* can be found in Chapter 12.

10.6 Limitations

Pearling has been designed to work for the extraction of thin vessels in both 2D and 3D images. If the desired application is to extract more complex features, such as entire organs in a 3D medical scan or a square parking-lot in a satellite image, then other approaches discussed above are more applicable.

For the case of 3D images, it is possible that a desired vessel for segmentation may not have a circular cross-section, but an elliptical or more complex cross-section instead. Because *pearling* uses a ball to trace vessels, it may produce undesirable

results if appropriate parameters are not set. In order to accommodate these more complex boundaries, the target ratio p used in Equation 32 can be set higher to allow the pearls to contain more "bad" pixels such that the irregular boundary is encompassed. In the examples presented here and in the next chapter, such a situation was not encountered.

10.7 User Control

The *pearling* process is initiated by the user manually selecting a point on the image and giving a direction vector, as well as identifying regions of "good" and "bad". In 2D, this is accomplished by directly clicking on the image, and in 3D, the user first selects a cross-section and proceeds as in 2D (Figures 91,92). An initial segmentation is then computed (Figure 93).

As *pearling* is computationally efficient, it affords the user interaction with data without delays. In Figure 88 (top) we show an example where an operator can select a string by simply moving the mouse over any pearl in the string. Once selected, the branch can then be deleted with the press of a key, as shown in Figure 88 (center). This process works the same way in 3D, the result of which is shown in Figure 94. The operator can also add strings to an incomplete segmentation by simply selecting a pearl and drawing a new direction vector from it, as shown in orange in Figure 88 (center) and similarly in 3D as shown in Figures 95, 96. The results of growing the new strings are shown in Figure 88 (bottom) and in Figure 97.

For more direct control in 2D, the operator may provide a rough trace of the centerline of a desired stroke, as shown in Figure 89 (orange). Pearls are placed at samples along the curve and converge in real-time thus forming the stroke (Fig. 89 (magenta)) as the operator is attempting to trace it with a stylus. In this mode, the operator uses a stylus to quickly trace a rough curve through the desired stroke structure and then as desired, adds branches. A preset mode or the stylus speed when

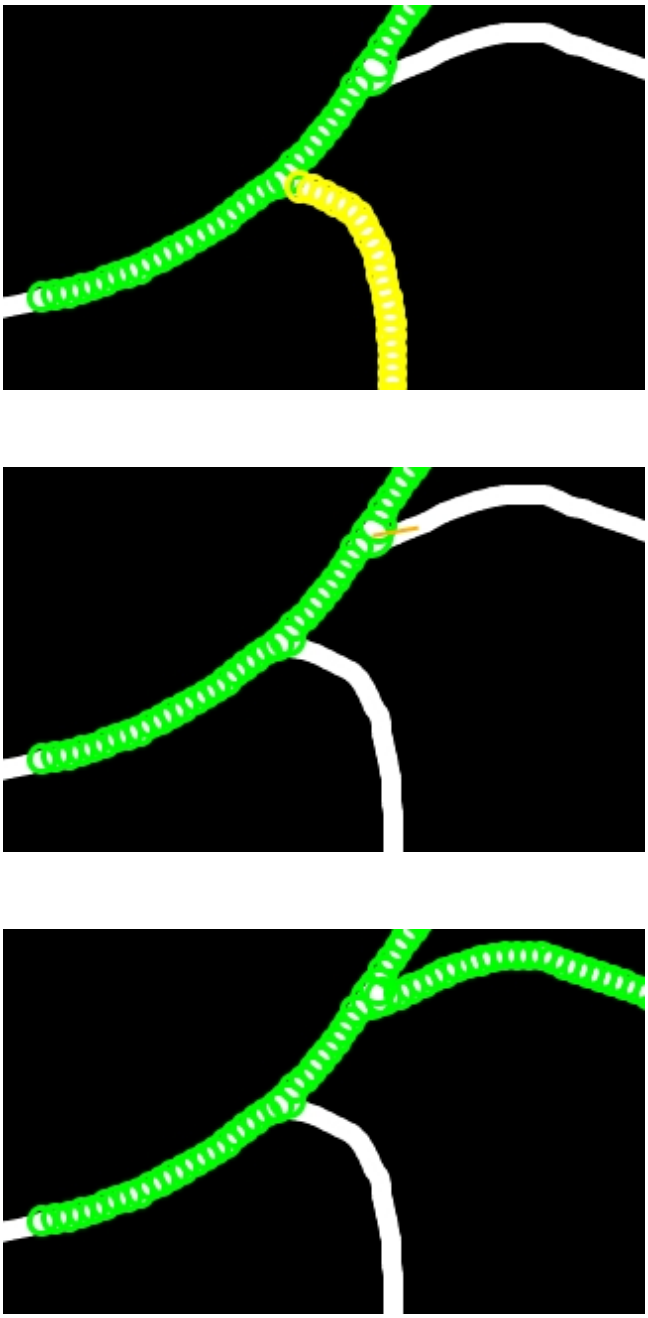


Figure 88: An initial incomplete *pearling*, with one string selected (yellow) by the operator (top). The result of the deleting the selected string and also the initial input for a new string (orange) by the operator (center), and the result (bottom).

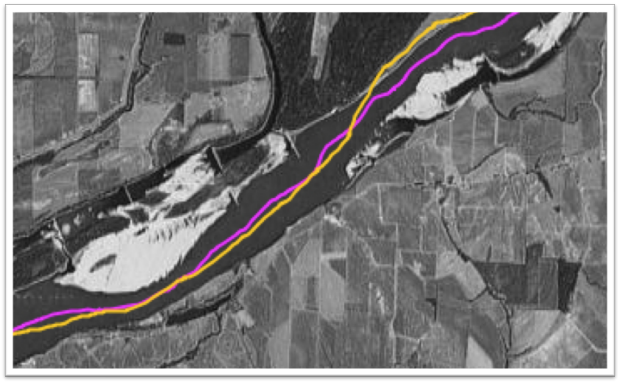


Figure 89: A rough trace of the centerline of the river supplied by the user (orange) has been interactively corrected (magenta).

the stylus is released indicate how far past the release point *pearling* will grow the stroke structure. As the operator is still tracing the curves, *pearling* computes the idealized strokes (centerline and radius) and displays them. Each new trace either adds or removes a branch.

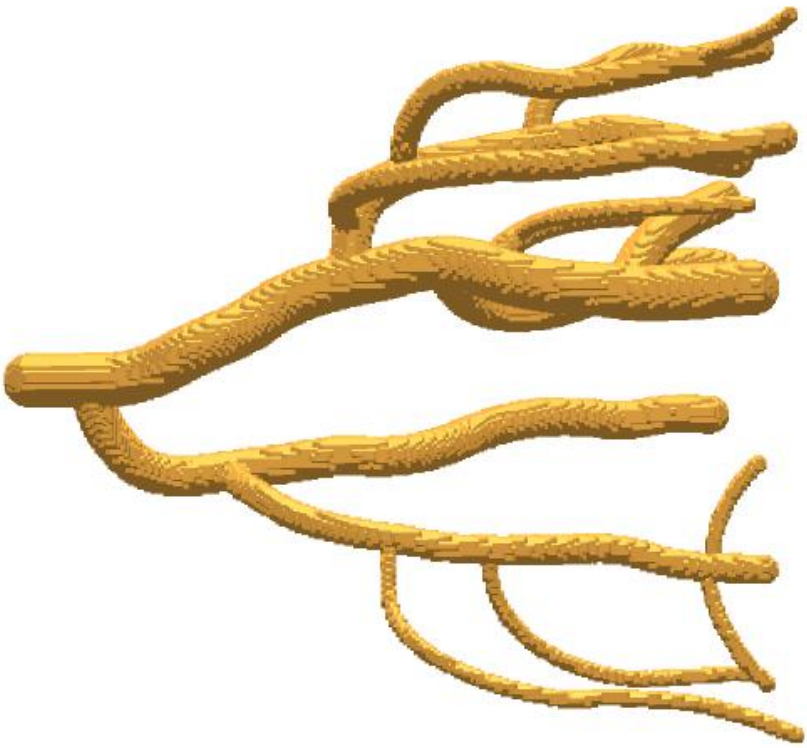


Figure 90: Segmentation obtained through global thresholding.

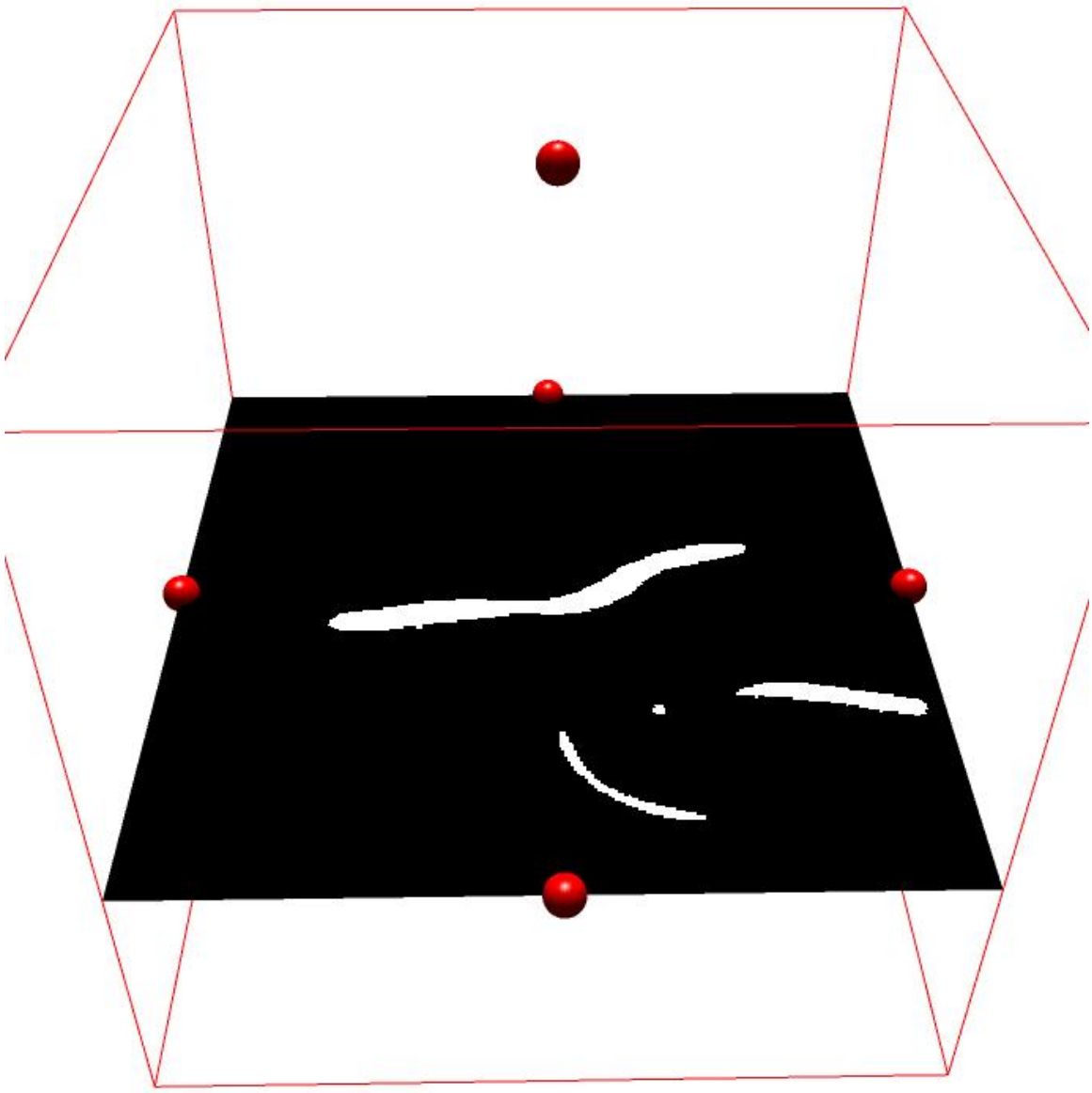


Figure 91: A horizontal slice through the volume.

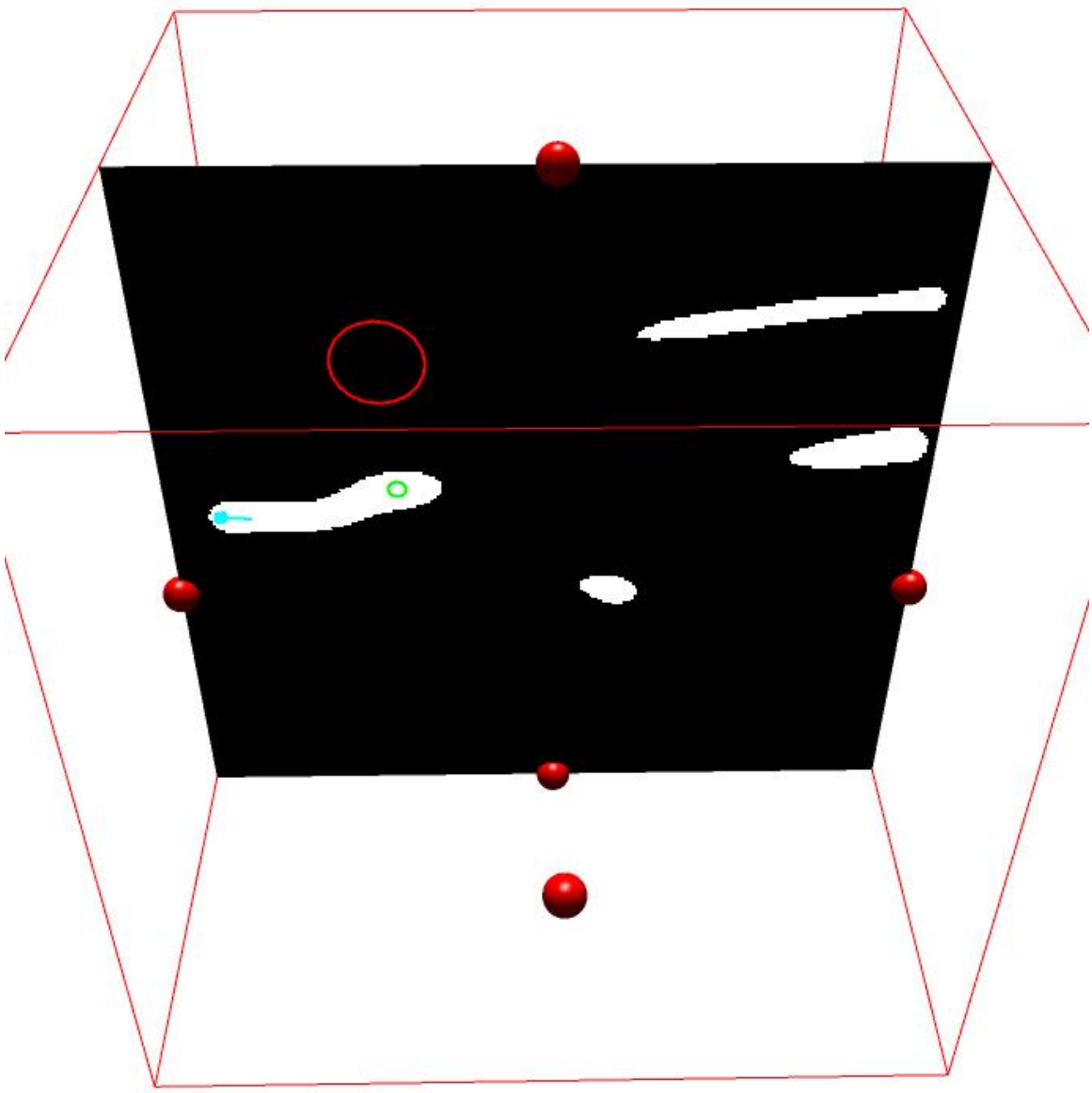


Figure 92: A different slice in which the operator has marked selected examples of good (green) and bad (red) material and has identified the seed and initial growth direction (cyan).

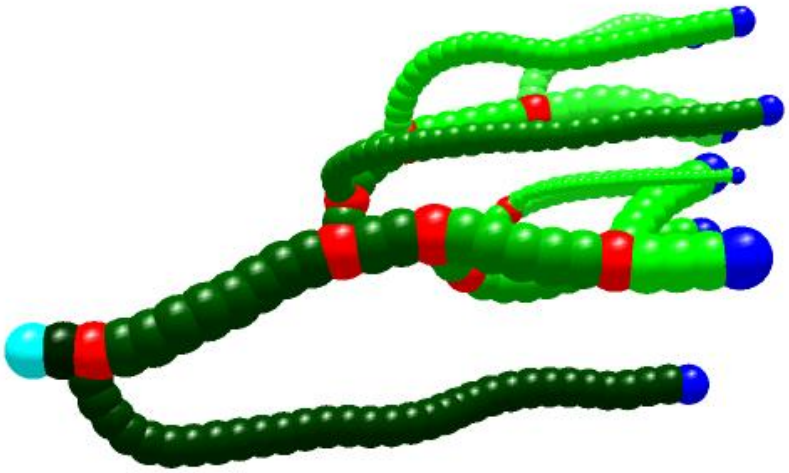


Figure 93: Initial tree extracted by *pearling* in 1.23 seconds with the seed-pearl (cyan), branching-pearls (red), leaf-pearls (blue), and control-pearls (green).

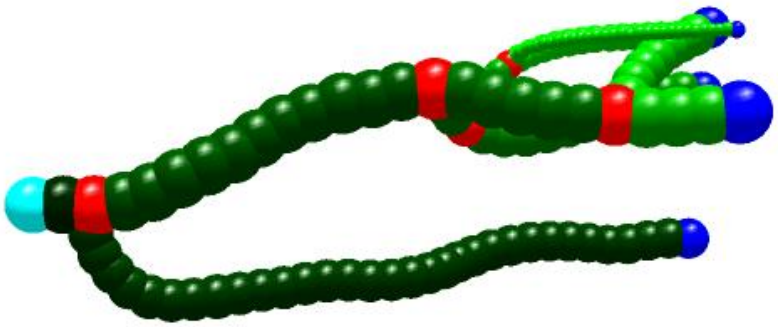


Figure 94: By clicking on the base of undesired branches, the operator has trimmed the tree.

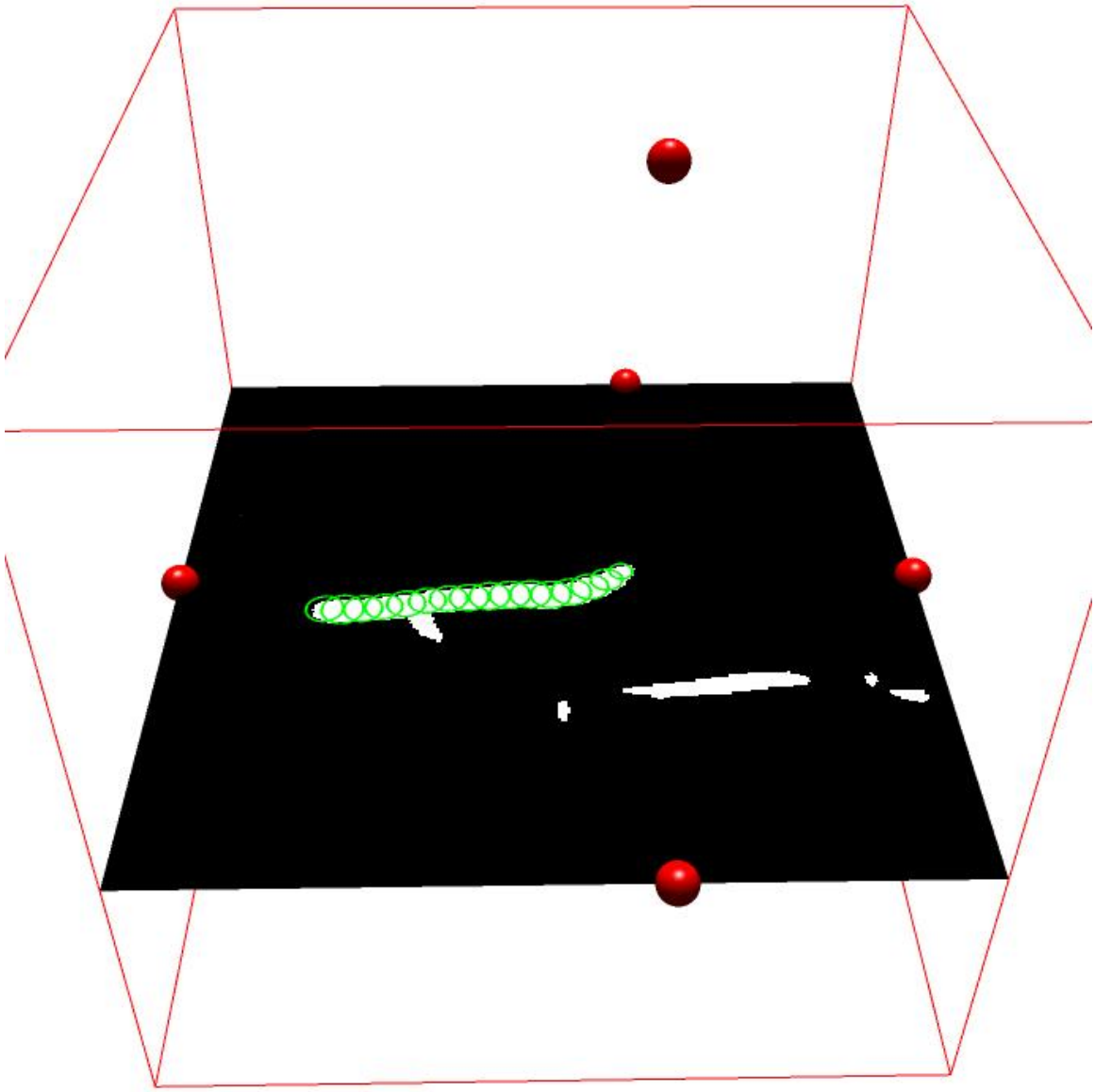


Figure 95: By sliding the cross-section along the spine, the operator has discovered a missing branch.

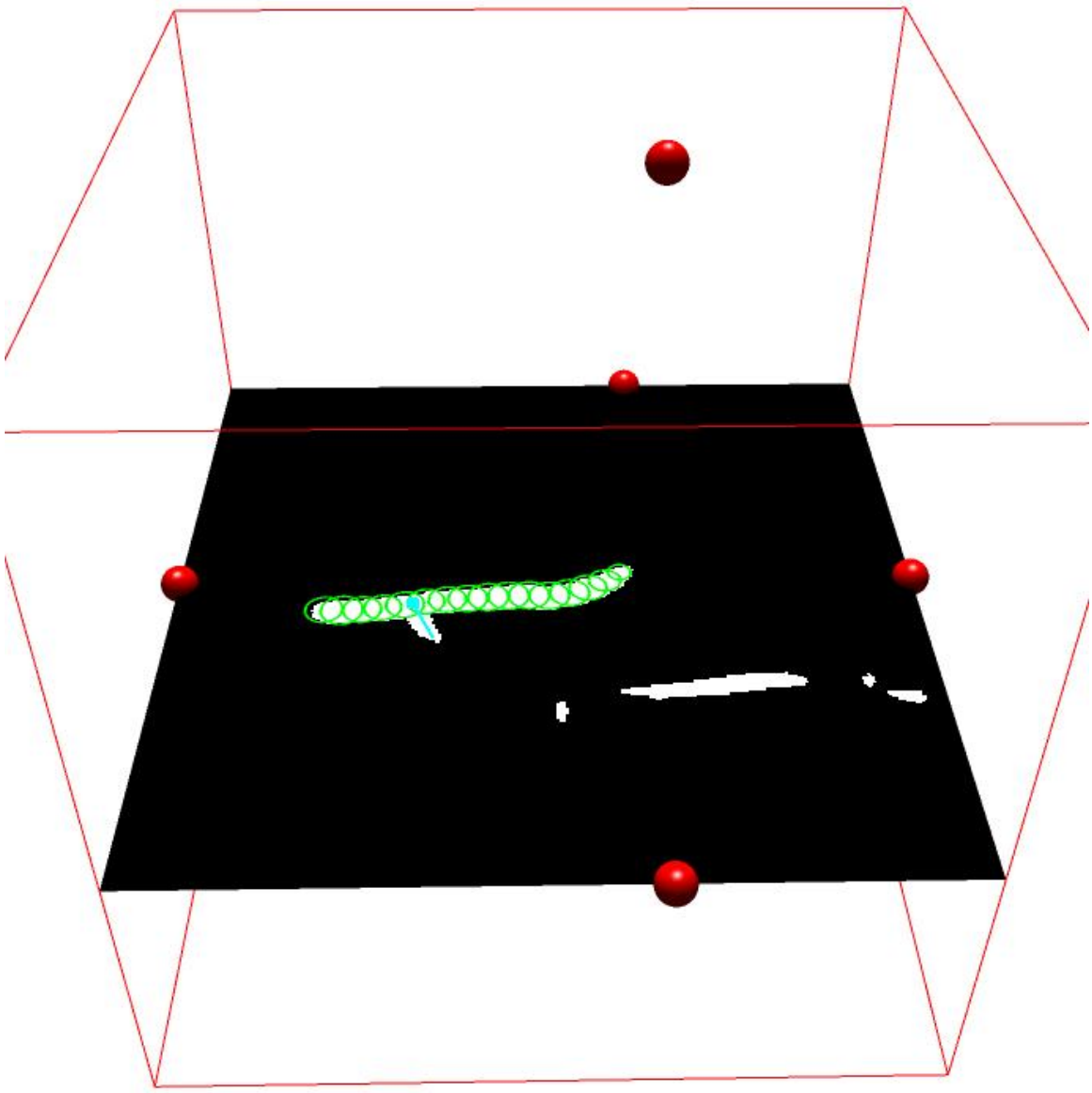


Figure 96: The direction of the desired branch is indicated in cyan.

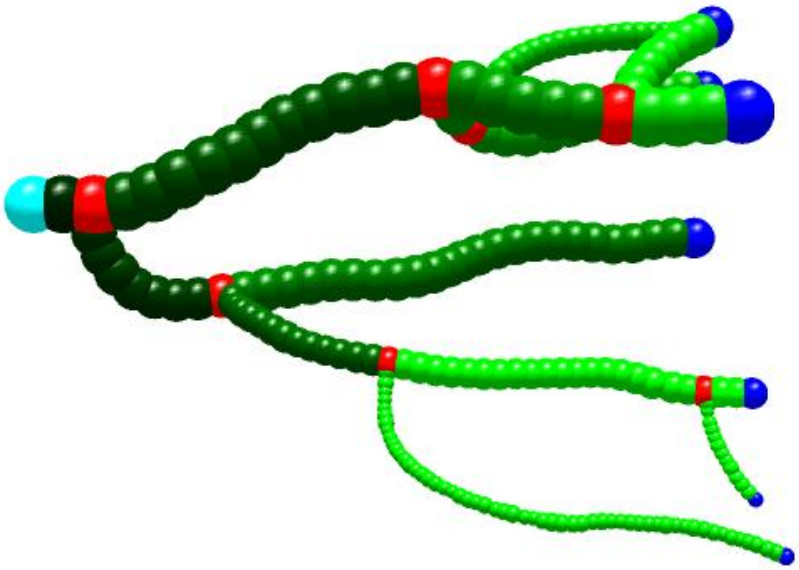


Figure 97: New portion added to the tree at the desired location.

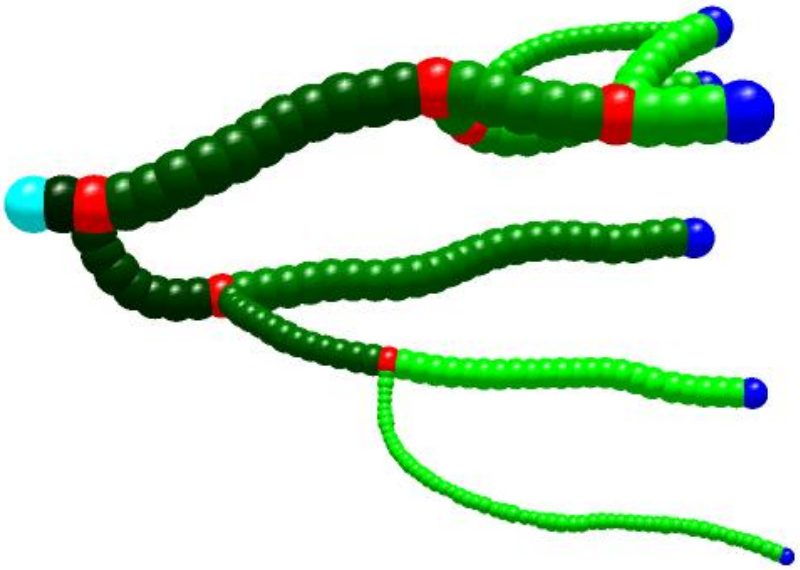


Figure 98: A small branch of the new portion removed.

CHAPTER XI

SKINNING AND FAIRING PEARL STRINGS

11.1 Building a continuous model from pearl strings

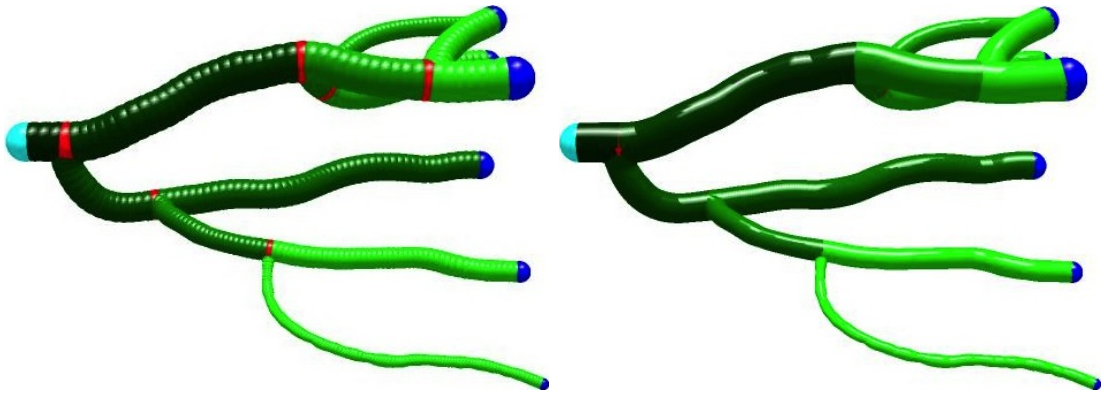


Figure 99: (Left) Branches after 1 refinement step. (Right) Branches after 5 refinements.

A continuous model W of each vessel is obtained by computing continuous functions $\mathbf{c}(t)$ and $r(t)$ that interpolate the discrete set of end, branching and contour pearls. We model W as the union of an infinite set of balls [81], which is and may be expressed in parametric form as $W = \mathbb{B}(\mathbf{c}(t), r(t))$ for $t \in [0, 1]$. If we assume that the pearls are more or less uniformly spaced along the vessel, we can define the string as the limit of a four-point subdivision process, which, at each iteration, introduces a new pearl between each pair of consecutive pearls as shown in Figure 99.

11.2 Sampling the pearl skin

Although producing a dense set of pearls may produce a smooth result for rendering (Figure 99 (right)), in order to manipulate the result using standard surface deformation techniques, or facilitate fluid simulations, a boundary representation must be obtained. Included here is an algorithm for computing an envelope for a single strand

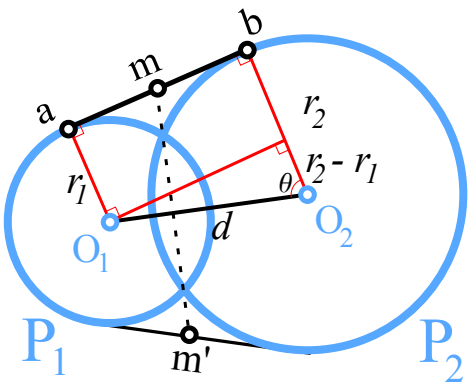


Figure 100: Construction for computing a midpoint (m) on the edge of the convex hull of two pearls.

of pearls for both 2D and 3D.

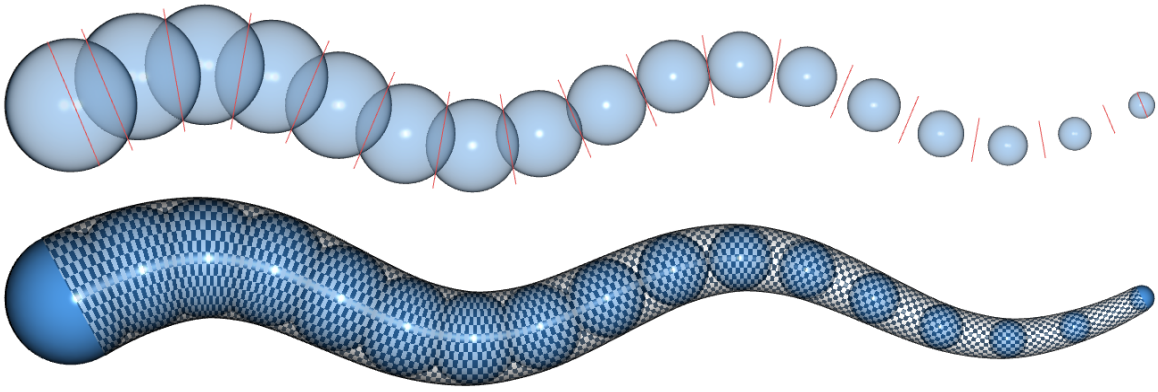


Figure 101: (Top) The cross-sections (red) computed between each pair of adjacent balls. (Bottom) The skin created by subdividing the input balls (blue) as a 4D curve and connecting each adjacent cross-section circle with a triangle-strip.

Once a dense sampling of pearls has been obtained through subdivision, the next step is to construct a series of cross-sections, one for each consecutive pair of pearls. This is done by computing samples (a pair of points in 2D, a circle in 3D) on the convex hull between two consecutive pearls. Let P_1 and P_2 be two consecutive pearls with centers $\mathbf{o}_1$, $\mathbf{o}_2$ and radii r_1, r_2 respectively, such that $r_2 > r_1$. Let d be the distance between $\mathbf{o}_1$ and $\mathbf{o}_2$. The angle θ (Fig. 100) is calculated as $cos^{-1}((r_2 - r_1)/d)$. The point $\mathbf{b}$ where an edge of the convex hull meets pearl P_2 is $\mathbf{b} = \mathbf{o}_2 + R(r_2 \cdot \frac{\overrightarrow{\mathbf{o_1 o_2}}}{|\overrightarrow{\mathbf{o_1 o_2}}|})$, where R is a rotation by angle θ. Similarly, the point a becomes $\mathbf{a} = \mathbf{o}_2 + R(r_1 \cdot \frac{\overrightarrow{\mathbf{o_1 o_2}}}{|\overrightarrow{\mathbf{o_1 o_2}}|})$. The

final cross-section point $\mathbf{m}$ is then $(\mathbf{a}+\mathbf{b})/2$. In 2D, the other cross-section point $\mathbf{m}'$ is obtained by reflecting $\mathbf{m}$ across $\overrightarrow{\mathbf{o_1 o_2}}$. In 3D, the cross-section circle is computed as the rotation of $\mathbf{m}$ around the axis $\overrightarrow{\mathbf{o_1 o_2}}$. Figure 101 (top) shows cross-sections computed for a string of balls.

Notice that if the cross-sections are connected, the edges will intersect the input balls. However, in the limit of subdivision by a four-point subdivision process (Fig. 99), a smooth skin can be obtained by connecting the densely sampled cross-sections with line segments (2D) or triangle strips (3D) (Figure 101 (bottom)).

11.3 Surface construction from circular contours

In order to construct a smooth tubular surface from a set of non-coplanar circles, as obtained from the above process, corresponding samples on consecutive circles must be computed. We propose the following simple solution.

The process above yields an ordered set of circles C_i with centers $\mathbf{o}_i$ and radii r_i; each on a plane with normal $\widehat{\mathbf{N}}_i$. Let C_0 be the first contour. We place m uniformly-spaced samples $\mathbf{c}_n$ on C_0. Let $\alpha_{0,1}$ be the angle between $\widehat{\mathbf{N}}_0$ and $\widehat{\mathbf{N}}_1$, $\alpha_{0,1} = cos^{-1}(\widehat{\mathbf{N}}_0 \cdot \widehat{\mathbf{N}}_1)$. Let $\overrightarrow{A}_{0,1}$ be the result of the vector cross product $\widehat{\mathbf{N}}_0 \times \widehat{\mathbf{N}}_1$. Let the difference between the centers of the circles be $D_{0,1} = \mathbf{o}_1 - \mathbf{o}_0$. For each sample $\mathbf{c}_n$ on C_0, we compute its corresponding point $\mathbf{c}'_n$ on C_1 as $R(\overrightarrow{A}_{0,1}, \alpha_{0,1})(r_1/r_0)(\mathbf{c}_n - \mathbf{o}_0) + D_{0,1} + \mathbf{o}_1$. This process is then repeated in order for all remaining circle-pairs C_i, C_{i+1}. Finally, for each circle-pair C_i, C_{i+1}, quads are computed between all points $\mathbf{c}_n, \mathbf{c}_{n+1}, \mathbf{c}'_{n+1}, \mathbf{c}'_n$ for $n = [0, m]$ and a mesh is defined.

CHAPTER XII

APPLICATIONS TO MEDICINE

The applications and results discussed here were motivated by and are the result of collaborations with Siemens Corporate Research (SCR) in the area of medical imaging.

12.1 Image Segmentation

We have evaluated the *pearling* approach, as discussed in Chapter 10, on a variety of medical image datasets, both 2D and 3D. Fig. 102 shows a segmentation of a MR angiographic image in 2D, with the pearls shown (right) and the approximate medial axis defined by the centers of the pearls (left).

In Fig. 103 (left), a thresholded volume rendering of the original angiography dataset is shown. Then, in Fig. 103 (center), we show the initial result of the *pearling* algorithm which contains 328 pearls and took 1.1 seconds. Finally, in Fig. 103 (right), we show the result of user editing where the goal was to trim everything away except the main vessel, the aneurysm, and the two major branches leading from that junction. This interaction took about 30 seconds total and involved nine deletions.

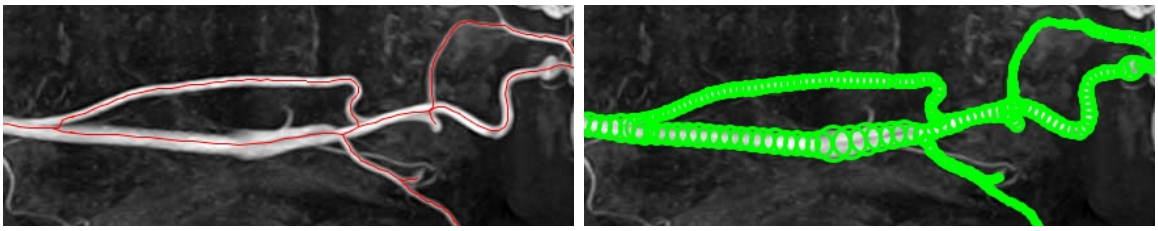


Figure 102: An MR angiographic image with centerlines (left) and discrete pearls (right) computed by *pearling* in 27ms on an image with size 436x168.

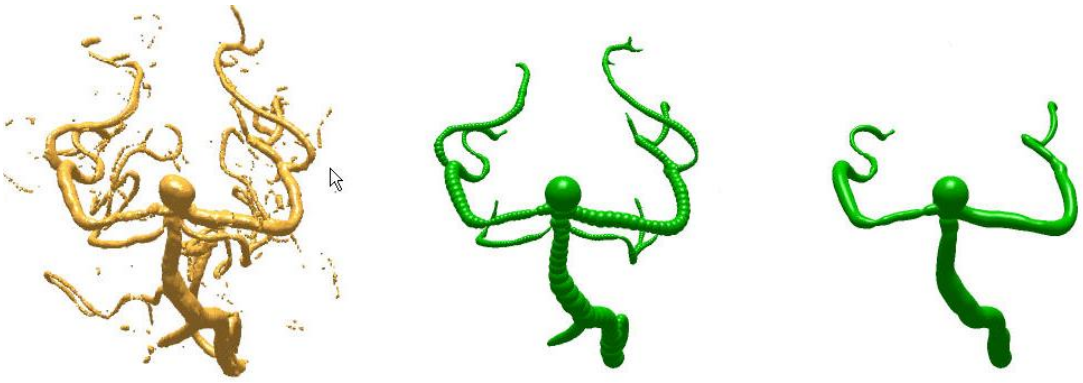


Figure 103: Left: Thresholded volume rendering of the original angiography dataset. Center: Initial result of the *pearling* algorithm (328 pearls, 1.1 seconds). Right: result of user editing and refinement (9 deletions, 30 seconds).

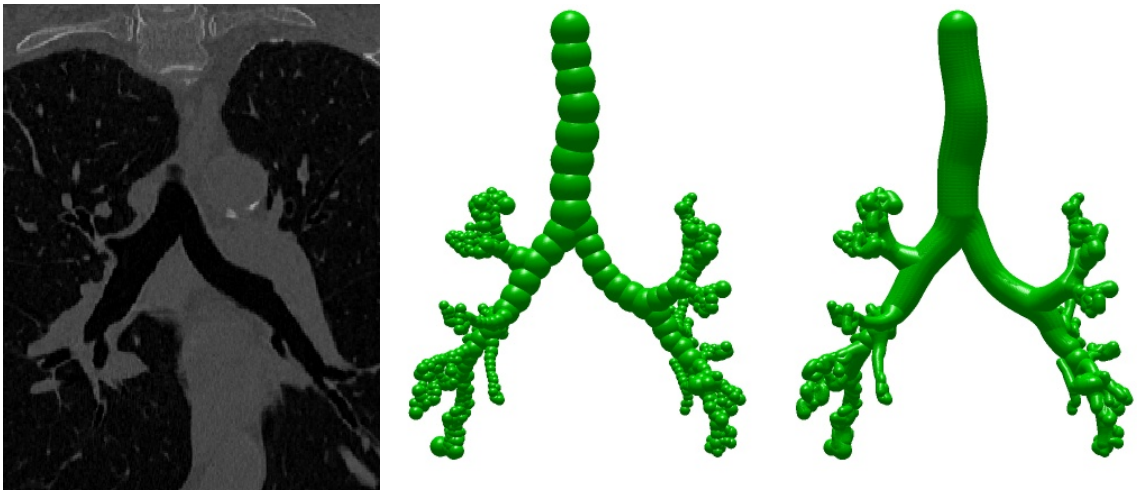


Figure 104: Left: Slice of a chest CT scan showing bronchial tubes. Center: Result of the initial run of *pearling* (638 pearls, 1.5 seconds). Right: Result of user editing and refinement rendered within the volume itself (2 deletions, 10 seconds).

Figure 105: Left: Detailed volume render of pulmonary arteries. Center: Initial *pearling* result. Right: Refined *pearling* result.

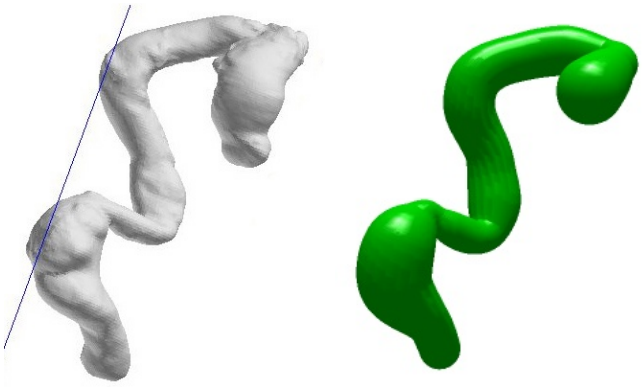


Figure 106: Left: Aorta segmention obtained via level set methods (10 minute execution on reduced volume). Right: *pearling* result using an endpoint and no trimming (1.5 second execution time on full-resolution image).

In Fig. 104 (left) a slice of the chest CT scan is shown within our *pearling* environment. Then, in Fig. 104 (center), we show the result of the initial run of *pearling*. This result contains 638 pearls and executed in 1.5 seconds. Even though there are almost twice as many pearls as the angiography, the running times are similar because most of the pearls in this example are extremely small and converge faster. Finally, in Fig. 104 (right), we show the result of refinement, producing a smooth structure.

The next example was chosen to show the result of *pearling* on a more detailed dataset. In Fig. 105 (left), a volume rendering of the dataset is shown, which includes the aorta, heart, kidneys and pulmonary arteries. In Fig. 105 (center), we show the result of *pearling*, which contains 10186 pearls and took 67 seconds to complete. Then, in Fig. 105 (right), the result of the refinement is shown, which creates smooth tubular structures.

In such a complex example, if the operator only needs to examine a subset of the image, an endpoint should be used to tell *pearling* to explicitly stop. The result in Fig. 106 (left) shows such an example of an aorta. No final trimming was needed for this result, and the execution took 1.5 seconds on the raw 512x512x459 volume. The result shown in Fig. 106 (right) is from a level-set approach, which required 10 minutes of running time on a reduced volume of size 256x256x229, because the full size volume combined with temporary structures wouldn't fit in the RAM of our test machine. Note that the *pearling* result is generated in over two orders of magnitude less time.

12.2 Future work on crust extraction

As discussed in Chapter 10, *pearling* provides an idealized approximation of the original image data. Image details which are important for a visual diagnosis by a doctor or radiologist may be lost in the process. Therefore, we introduced the concept of the crust, which identifies a region near the boundary of the *pearling* segmentation. This

region contains important image data, such as the arterial wall of a segmented artery. We propose only storing the crust and *pearling* segmentation to reduce storage and transmission costs and also to allow for efficient rendering of only the desired portions of the original volumetric data.

12.3 Future work on slice interpolation

The problem (encountered in the segmentation of medical scans) of constructing a surface in 3D that interpolates between each pair of consecutive planar *cross-sections* may be solved [27] using the morphing between the projection, onto the same plane, of the two cross-section curves. This problem of surface reconstruction has been studied extensively [29][9][1][47][32].

12.3.1 Limitations

Our investigation of the benefit of *b-morphs* in Chapters 7, 8 to the problem of cross-section interpolation has limitations:

1. We assumed *b-compatibility*

2. We considered only two consecutive slices instead of building a smooth surface through the whole series, as proposed in [10]

The first limitation may be addressed using the composite *b-morph* as discussed in Chapter 9. Because the *b-morph* reaches the interpolated contours at right angles, the projection of these trajectories on the slice plane all meet tangentially with angle 0 or π. We expect that this property may help researchers devise solutions that smoothly connect surface sections generated by *b-morphs*, addressing the second limitation. Furthermore, the *b-morph* as discussed here is not suited for dealing with topological changes, as discussed for example in [48].

CHAPTER XIII

APPLICATIONS TO ANIMATION

While some of the underlying mechanisms have benefited from the advent of digital technology, a 2D production today follows much the same basic workflow as traditional animation [62]. The process begins with a storyboard, which provides a visual representation of the story. In layout, the staging for each scene is designed, including establishing the setting, choosing and placing character and prop elements, and specifying camera motion and cuts.

Animators first produce the subset of rough drawings that lay down the core of the action, the "keys." A "clean up" artist is then responsible for taking the rough key drawings and producing clean lines that remain true to the original intent. Each key drawing has one or more associated timing charts in order to specify how many drawings should be produced between the keys, and at what time intervals. An inbetweening artist then has the job of drawing the intermediate frames in order to produce a seamless motion.

The inbetweening process for similar key drawings can be very tedious for the artists due to the accuracy required and lack of artistic interpretation. Thus, the problem is ideal for automation and although there has been over 40 years of research devoted to this task [79], the problem remains unsolved.

Some of the basic algorithmic pieces necessary for a stroke-based approach are:

1. Vectorization of the cleaned-up drawings

2. Segmentation into a graph of connected strokes

3. Stroke-to-stroke correspondence for consecutive keys, which may have different

topologies

4. Stroke-to-stroke interpolation

We discuss in this chapter how *pearling* can be used to vectorize drawings and how the *b-morph* can be applied to stroke interpolation. This work is motivated by and the result of a collaboration with Walt Disney Animation Studios.

13.1 Future work on vectorization

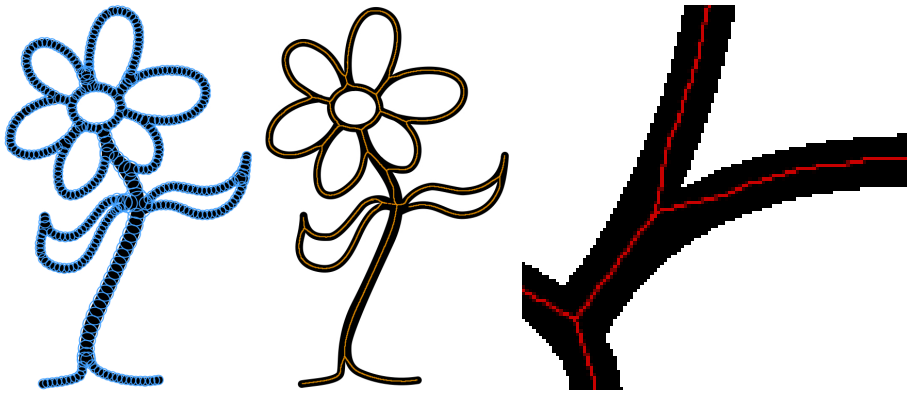


Figure 107: Left: Pearls computed automatically for the drawing of a flower. Center: The curves created by joining the centers of the pearls. Right: A zoomed in section of the center lines. Notice that the obtained medial axis does not correspond to the trajectory of the artist's pen.

Before any shape processing algorithms can be applied to traditional animation, the geometry of the artists' drawings must be captured. Because artists often prefer working with pencil and paper, vectorization is necessary in order to create digital representations of the drawings. Such drawings are likely to contain variable thickness strokes, and sometimes different levels of shading. For this reason, *pearling* (Chapter 10) is an ideal candidate. Figure 107-left shows the result of *pearling* segmentation for a simple drawing. In Figure 107-center, the centers of the pearls are connected to produce an approximation of the medial axis of the drawing.

Notice, however, at the bifurcations of this medial axis, it is not a reasonable representation of the centerlines actually drawn by the artist's pen (Figure 107-right).

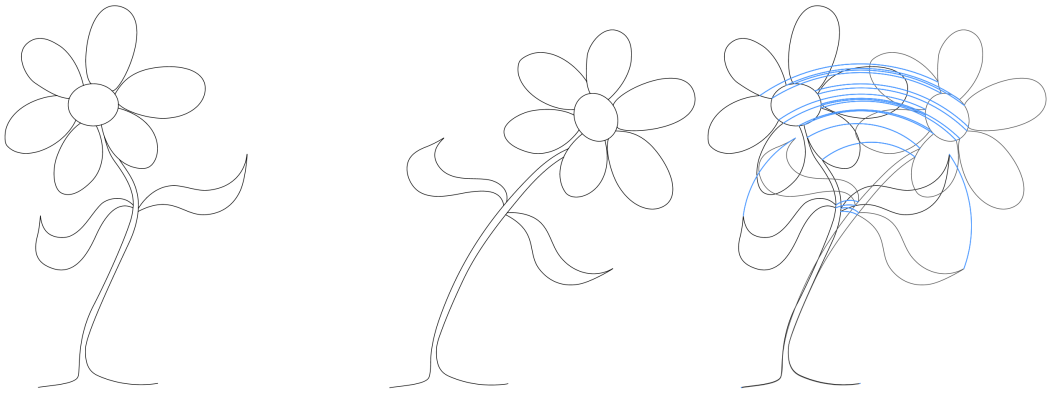


Figure 108: (Left and Center) Two keyframes of an animation. (Right) Trajectories (blue) that define the desired motion have been specified between each pair of corresponding points on the 2 keyframes.

This will affect a rendering of the curves based on the *pearling* segmentation and produce unexpected artifacts once geometric operations are applied to the geometry. Future work is necessary to ensure *pearling* accurately vectorizes the strokes at the junctions.

13.2 Future work on morphing (inbetweening)

Morphing is a fundamental tool in animation design where *in-between* [21] frames are produced from a sparse set of key-frames that are often drawn [62] digitally or have been vectorized from pencil and paper drawings. Although several attempts at automating the construction of in-between frames have been developed [65], the artist responsible for in-betweening likes to have control over correspondence and the trajectories for selected landmarks or stroke end-points. These specifications are difficult to fully automate because they involve aesthetic judgement, style guidelines, and context semantics about the relative 3D motions of the strokes and their mutual occlusions.

Once these matching and control trajectories are given, as in Figure 108, the overall problem is naturally broken into a series of tight in-betweening tasks [91]. These are viewed as tedious and hence are a prime candidate for artist-supervised

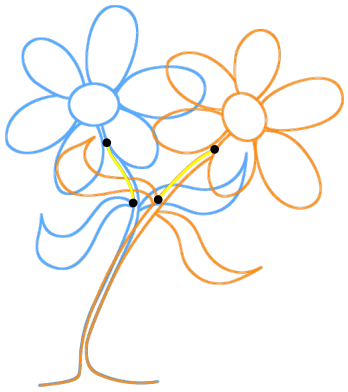


Figure 109: Two corresponding strokes from the two keyframes are highlighted in yellow.

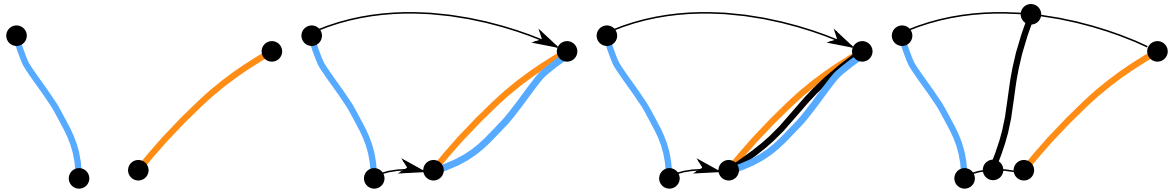


Figure 110: (a) Two corresponding strokes from Figure 109. (b) The blue curve is transformed to bring it into endpoint alignment with the orange curve. (c) An inbetween stroke is computed. (d) The inbetween curve is transformed to have its endpoints aligned to the proper location along the specified trajectories (Figure 108).

automation. In most of such tight in-between tasks, the goal is to generate a number of intermediate frames between two reasonably simple and similar curve segments, such as the two highlighted in Figure 109.

This task can be accomplished by applying a morph to each pair of corresponding curves. For example, the two curves from Figure 109 can be brought into endpoint alignment by applying an affine transformation (Fig. 110 (b)). Once the curves have matching endpoints, an inbetween, or morph, curve is computed for every time t as desired by the artist (Fig. 110 (c)). Another affine transformation is then applied to each resulting inbetween curve in order to put its endpoints at the appropriate point in time along the defined trajectories (Fig. 110 (d)).

13.3 B-morph motivation and limitations

At this point, it is unreasonable to ask the artist to identify the best candidate techniques that promise to generate acceptable morphs. Then, rather than offloading upon the artist the burden of choosing the best one in each case, one may want to compare these techniques to better assess the strength of each. The comparisons in Chapter 8 are a modest–although we hope useful–step in this direction. It may not be the final answer to tight in-betweening for several reasons: (1) the quantitative quality measures that we use may not reflect artistic concerns. (2) for practical reasons, we compare the proposed *b-morphs* to our simple and un-optimized implementations of candidate techniques, and not to state of the art solutions. (3) we do not take into account the broader context of the whole animation, but instead focus on interpolating only the instances of the same stroke in two consecutive key-frames. Nevertheless, we feel that the experiments described in this thesis are useful and that the conclusions we draw from them about the specific benefits of the *b-morphs* will help the reader appreciate their potential.

In the application of animation, the quality of the morph is important as one typically favors a solution where the animation is smooth and free from self-intersections [40] and of unnecessary distortions. We have shown that when the curves are *b-compatible*, the *b-morph* always satisfies these properties.

However, there may be cases when an artist desires specific points on each curve to correspond. Likewise, an artist may not want the intersection points of two intersecting curves to remain fixed throughout the morph. For these cases, the *b-morph* may not be the best candidate for producing inbetween curves.

CHAPTER XIV

CONCLUSIONS

In this thesis, we have introduced several *tangent-ball*-based operators where a ball is placed such that it has tangential contact with two or more points of either one or two sets. Because of the properties of a ball, when it is placed tangent to multiple points, its center lies on the medial axis of the complement of the tangent set(s). We have shown how to exploit this property for the blending of shapes, morphing of shapes, and segmenting of images.

We have included the implementations of these operators and discussed their usefulness in applications of shape processing. We have highlighted the specific applications of these tools in the fields of medical imaging as well as computer animation.

14.1 Relative Blending

The contributions reported in Chapter 4 include the invention of the concept of *relative blending* and a set theoretic formulation of it. We outlined practical approaches for implementing *relative blending* and proposed a novel interface for specifying the bounding shape that controls the local radius. We discussed several applications of relative blending, including interactive design and shape comparison. We hope that these initial contributions will fuel further research on efficient algorithms for computing *relative blendings* and inspire new applications.

14.2 Ball-morph

We have proposed a family of morphs between curves which are *b-compatible* in Chapter 7. All are based on variations of the medial axis construction. We have compared them to one another and to several other simple morphs in Chapter 8.

We used four measures of morph quality in our comparison, as well as three surface measures for comparing them as surface reconstruction techniques.

We conclude that for the cases of *b-compatible* shapes, the *b-morphs* offer a precise and desirable result in terms of *distortion*, *travel distance*, as well as curvature. The *Circular b-morph* is guaranteed to produce morph curves of C^{k-1} continuity for input curves of C^k for $k \geq 2$ [24].

When curves are not *b-compatible*, we have shown in Chapter 9 that the *b-morph* may be extended to produce morphs by applying recursive *relative blending* operations. This method supplies a fast result comparable to that produced by the *Heat Propagation* morph in these non-*b-compatible* cases.

14.3 Pearling

In Chapter 10 we presented *pearling*, a new method for semi-automatic segmentation and geometric modeling of tube-like structures. *Pearling* performs a segmentation by computing an ordered series of pearls that discretely model the tubular structure geometry. Subdivision is used to define a continuous model. The computational efficiency of *pearling* affords efficient user interaction with the segmentation, allowing the operator to correct for errant segmentation results or guide the segmentation in a particular direction through the data.

Sometimes the idealized *pearling* result doesn't contain enough information to perform a more precise analysis, so we presented the concept of a crust around the border region. Important image information, such as the inner wall of an artery, is identified in the crust region and can be used to support a more detailed analysis.

While more comprehensive validation of the algorithm is required, from our experimental results we conclude that *pearling* results in highly efficient segmentation

of tubular structures in both 2D and 3D images, and holds much promise for semi-automatic image segmentation, especially in the application of medical image segmentation, as demonstrated in Chapter 12.

REFERENCES

[1] Akkouche, S. and Galin, E., "Implicit surface reconstruction from contours," *Vis. Comput.*, vol. 20, no. 6, pp. 392–401, 2004.

[2] Alexa, M., "Differential coordinates for local mesh morphing and deformation," *The Visual Computer*, vol. 19, pp. 105–114, 2003.

[3] Alexa, M., Cohen-Or, D., and Levin, D., "As-rigid-as-possible shape interpolation," in *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 157–164, ACM Press/Addison-Wesley Publishing Co., 2000.

[4] Alt, H., Knauer, C., and Wenk, C., "Bounding the Fréchet distance by the Hausdorff distance," in *Proc. 17th European Workshop on Computational Geometry*, pp. 166–169, 2001.

[5] Aronov, B., Seidel, R., and Souvaine, D., "On compatible triangulations of simple polygons," *Comput. Geom. Theory Appl.*, vol. 3, no. 1, pp. 27–35, 1993.

[6] Asada, H. and Brady, M., "The curvature primal sketch," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 1, pp. 2–14, 1986.

[7] Atallah, M., "A linear time algorithm for the Hausdorff distance between convex polygons," *Information Processing Letters*, vol. 16, pp. 207–209, 1983.

[8] Badris, L. and Patrikalakis, N. M., "Blending rational b-spline surfaces," in *Proceedings of Eurographics '89* (Hansmann, W., Hopgood, F. R. A., and Strasser, W., eds.), pp. 453–462, 1989.

[9] Barequet, G. and Sharir, M., "Piecewise-linear interpolation between polygonal slices," *Comput. Vis. Image Underst.*, vol. 63, no. 2, pp. 251–272, 1996.

[10] Barequet, G. and Vaxman, A., "Nonlinear interpolation between slices," in *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, (New York, NY, USA), pp. 97–107, ACM, 2007.

[11] Barr, A. H., "Global and local deformations of solid primitives," in *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 21–30, ACM, 1984.

[12] Beier, T. and Neely, S., "Feature-based image metamorphosis," *SIGGRAPH Computer Graphics*, vol. 26, no. 2, pp. 35–42, 1992.

[13] Besl, P. and McKay, N., "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[14] Blackmore, D., Leu, M. C., Wang, L. P., and Jiang, H., "Swept volume: a retrospective and prospective view," *Neural, Parallel Sci. Comput.*, vol. 5, no. 1-2, pp. 81–102, 1997.

[15] Bloomenthal, J. and Shoemake, K., "Convolution surfaces," *SIGGRAPH Comput. Graph.*, vol. 25, no. 4, pp. 251–256, 1991.

[16] Bloor, M. I. G. and Wilson, M. J., "Spectral approximations to pde surfaces," *Computer-Aided Design*, vol. 28, pp. 145–152, 1996.

[17] Blum, H., "A Transformation for Extracting New Descriptors of Shape," in *Models for the Perception of Speech and Visual Form* (Wathen-Dunn, W., ed.), pp. 362–380, Cambridge: MIT Press, 1967.

[18] Braid, I., "Non-local blending of boundary models," *Computed-Aided Design*, vol. 29, pp. 98–100, 1997.

[19] Breen, D. E., "Constructive cubes: CSG evaluation for display using discrete 3d scalar data sets," in *Proceedings of Eurographics 91* (Purgathofer, W., ed.), pp. 127–142, 1991.

[20] Cao, L., Jia, Z., and Liu, J., "Computation of medial axis and offset curves of curved boundaries in planar domains based on the Cesaro's approach," *Computer Aided Geometric Design*, vol. 26, no. 4, pp. 444 – 454, 2009. Geometric Modeling and Processing 2008, 5th International Conference on Geometric Modeling and Processing.

[21] Catmull, E., "The problems of computer-assisted animation," *SIGGRAPH Comput. Graph.*, vol. 12, no. 3, pp. 348–353, 1978.

[22] Chandru, V., Dutta, D., and Hoffman, C., "Variable radius blending using Dupin cyclides," in *Geometric Modeling for Product Engineering* (Preiss, J. K. and Wozny, M., eds.), pp. 39–58, 1990.

[23] Chandru, V., Dutta, D., and Hoffmann, C. M., "On the geometry of Dupin cyclides," *The Visual Computer*, vol. 5, pp. 277–290, 1989.

[24] Chazal, F., Lieutier, A., Rossignac, J., and Whited, B., "Ball-map: Homeomorphism between compatible surfaces," *International Journal of Computational Geometry and Applications*, 2009 (to appear).

[25] Chazel, F., Lieutier, A., and Rossignac, J., "Orthomap: Homeomorphism-guaranteeing normal-projection map between surfaces," Tech. Rep. GIT-GVU-04-28, Georgia Institute of Technology, Graphics, Visibility, and Usability Center, 2004.

[26] Che, W., Yang, X., and Wang, G., "Skeleton-driven 2D distance field metamorphosis using intrinsic shape parameters," *Graphical Models*, vol. 66, no. 2, pp. 102–126, 2004.

[27] Chen, S. E. and Parent, R. E., "Shape averaging and its applications to industrial design," *IEEE Comput. Graph. Appl.*, vol. 9, no. 1, pp. 47–54, 1989.

[28] Chen, X. and Hoffmann, C. M., "Trimming and closure of constrained surfaces," tech. rep., Computer Science Department, Purdue University, 1993.

[29] Cheng, S.-W. and Dey, T. K., "Improved constructions of Delaunay based contour surfaces," in *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, (New York, NY, USA), pp. 322–323, ACM, 1999.

[30] Chuang, Lin, and Hwang, "Variable-radius blending of parametric surfaces," *The Visual Computer*, vol. 11, pp. 513–525, 1995.

[31] Cohen, L. D. and Kimmel, R., "Global minimum for active contours models: A minimal path approach," *International Journal of Computer Vision*, vol. 24, no. 1, pp. 57–78, 1997.

[32] Cohen-Or, D., Solomovic, A., and Levin, D., "Three-dimensional distance field metamorphosis," *ACM Transaction on Graphics*, vol. 17, no. 2, pp. 116–141, 1998.

[33] Cong, G. and Parvin, B., "A new regularized approach for contour morphing," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 1, pp. 458–463 vol.1, 2000.

[34] Cornea, N., Silver, D., Yuan, X., and Balasubramanian, R., "Computing hierarchical curve-skeletons of 3D objects," *The Visual Computer*, vol. 21, no. 11, pp. 945–955, 2005.

[35] Costa, L. and Cesar, R., *Shape Analysis and Classification*. CRC Press, 2001.

[36] Cui, M., Femiani, J., Hu, J., Wonka, P., and Razdan, A., "Curve matching for open 2D curves," *Pattern Recognition Letters*, vol. 30, no. 1, pp. 1–10, 2009.

[37] Dekkers, D., van Overveld, K., and Golsteijn, R., "Combining CSG modeling with soft blending using Lipschitz-based implicit surfaces," *The Visual Computer*, vol. 20, no. 6, pp. 380–391, 2004.

[38] Du, H. and Qin, H., "Medial axis extraction and shape manipulation of solid objects using parabolic PDEs," in *SM '04: Proceedings of the ninth ACM symposium on Solid modeling and applications*, (Aire-la-Ville, Switzerland, Switzerland), pp. 25–35, Eurographics Association, 2004.

[39] Dupin, C., *Applications de Gèomètrie et de Mecànique*. Paris: Bachelier, 1822.

[40] Efrat, A., Har-Peled, S., Guibas, L. J., and Murali, T. M., "Morphing between polylines," in *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, (Philadelphia, PA, USA), pp. 680–689, Society for Industrial and Applied Mathematics, 2001.

[41] Federer, H., *Geometric Measure Theory*. Springer Verlag, 1969.

[42] Filkins, P. C., Tuohy, S. T., and Patrikalakis, N. M., "Computational methods for blending surface approximation," *Engineering with Computers*, vol. 9, 1993.

[43] Fjällström, P. O., "Smoothing of polyhedral models," in *SCG '86: Proceedings of the second annual symposium on Computational geometry*, (New York, NY, USA), pp. 226–235, ACM, 1986.

[44] Fortune, S., "A sweepline algorithm for Voronoi diagrams," in *SCG '86: Proceedings of the Second Annual Symposium on Computational Geometry*, (New York, NY, USA), pp. 313–322, ACM, 1986.

[45] Foskey, M., Lin, M., and Manocha, D., "Efficient computation of a simplified medial axis," in *ACM Symposium on Solid Modeling and Applications*, pp. 96–107, 2003.

[46] Fu, H., Tai, C.-L., and Au, O. K., "Morphing with Laplacian coordinates and spatial temporal texture," in *Proc. Pacific Graphics '05*, pp. 100–102, 2005.

[47] Fuchs, H., Kedem, Z. M., and Uselton, S. P., "Optimal surface reconstruction from planar contours," *Commun. ACM*, vol. 20, no. 10, pp. 693–702, 1977.

[48] Gabrielides, N. C., Ginnis, A. I., Kaklis, P. D., and Karavelas, M. I., "G1-smooth branching surface construction from cross sections," *Computer Aided Design*, vol. 39, no. 8, pp. 639–651, 2007.

[49] Gelfand, N., Mitra, N. J., Guibas, L. J., and Pottmann, H., "Robust global registration," in *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing*, (Aire-la-Ville, Switzerland, Switzerland), p. 197, Eurographics Association, 2005.

[50] Gisch, D. and Ribando, J. M., "Apollonius' problem: A study of solutions and their connections," *American Journal of Undergraduate Research*, vol. 3, no. 1, pp. 15–25, 2004.

[51] Gomes, J., Darsa, L., Costa, B., and Velho, L., *Warping and morphing of graphical objects*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[52] Guo, H., Fu, X., Chen, F., Yang, H., Wang, Y., and Li, H., "As-rigid-as-possible shape deformation and interpolation," *J. Vis. Comun. Image Represent.*, vol. 19, no. 4, pp. 245–255, 2008.

[53] Guthe, M., Borodin, P., and Klein, R., "Fast and accurate Hausdorff distance calculation between meshes," *Journal of WSCG*, vol. 13, pp. 41–48, February 2005.

[54] Guy, A. and Wyvill, B., "Controlled blending for implicit surfaces," in *Proc. Implicit Surfaces '95–First Eurographics Workshop Implicit Surfaces*, (Grenoble, France), pp. 107–112, 1995.

[55] Hilbert, D. and Cohn-Vossen, S., *Geometry and the Imagination*. New York: Chelsea Publishing Co., 1952.

[56] Hoffmann, C. and Hopcroft, J., "Automatic surface generation in computer aided design," *The Visual Computer*, vol. 1, pp. 92–100, 1985.

[57] Hoffmann, C. and Hopcroft, J., *Geometric Modeling: Algorithms and New Trends*, ch. The Potential Method for Blending Surfaces and Corners, pp. 347–365. Philadelphia, PA: Farin G. E. eds., SIAM, 1987.

[58] Hoffmann, C. and Vanecek, G., "Fundamental techniques for geometric and solid modeling," in *Advances in Control and Dynamics* (Leondes, C., ed.), vol. 48, pp. 101–165, Academic Press, 1992.

[59] Hoffmann, C. and Hopcroft, J., "Quadratic blending surfaces," *Computer Aided Design*, vol. 18, no. 6, pp. 301–306, 1986.

[60] Hoffmann, C. M., *Geometric and solid modeling: an introduction*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1989.

[61] Hoffmann, C., "A dimensionality paradigm for surface interrogations," tech. rep., Cornell University, Ithaca, NY, USA, 1988.

[62] Johnston, O. and Thomas, F., *The Illusion of Life*. Disney Press, 1995.

[63] Kirbas, C. and Quek, F., "A Review of Vessel Extraction Techniques and Algorithms," *ACM Computing Surveys*, vol. 36, no. 2, pp. 81–121, 2004.

[64] Klein, R., Schilling, A., and Straβer, W., "Reconstruction and simplification of surfaces from contours," in *PG '99: Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, (Washington, DC, USA), p. 198, IEEE Computer Society, 1999.

[65] Kort, A., "Computer aided inbetweening," in *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, (New York, NY, USA), pp. 125–132, ACM, 2002.

[66] Krasauskas, R., "Branching blend of natural quadrics based on surfaces with rational offsets," *Computed Aided Geometric Design*, vol. 25, pp. 332–341, 2008.

[67] Kunkel, P., "The tangency problem of apollonius: three looks," *BSHM Bulletin: Journal of the British Society for the History of Mathematics*, vol. 22, no. 1, pp. 34–46, 2007.

[68] Leyton, M., *Symmetry, Causality, Mind*. MIT Press, 1992.

[69] Li, H. and Yezzi, A., "Vessels as 4d curves: Global minimal 4d paths to extract 3d tubular surfaces," in *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, (Washington, DC, USA), p. 82, IEEE Computer Society, 2006.

[70] Lien, J.-M., *Approximate convex decomposition and its applications*. PhD thesis, Texas A & M, College Station, TX, USA, 2006. Adviser-Amato, Nancy M.

[71] Llamas, I., "Real-time voxelization of triangle meshes on the GPU," in *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, (New York, NY, USA), ACM, 2007.

[72] Llamas, I., Kim, B., Gargus, J., Rossignac, J., and Shaw, C. D., "Twister: a space-warp operator for the two-handed editing of 3D shapes," in *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, (New York, NY, USA), pp. 663–668, ACM, 2003.

[73] Lorigo, L. M., Faugeras, O. D., Grimson, W. E. L., Keriven, R., Kikinis, R., Nabavi, A., and Westin, C.-F., "CURVES: Curve Evolution for Vessel Segmentation," *Medical Image Analysis*, vol. 5, pp. 195–206, 2001.

[74] Martin, R. R., "Principal patches - a new class of surface patch," in *Proc Eurographics 1983*, pp. 47–55, 1983.

[75] Martin, R. R., de Pont, J. J., and Sharrock, T., "Cyclide surfaces for computer-aided design," in *Mathematics of Surfaces* (Gregory, J., ed.), (Oxford), pp. 253–267, Oxford Univ Press, 1986.

[76] Matheron, G., *Eléments pour une thérie des milieux poreoux*. Paris: MassonParis, 1967.

[77] Medek, P., Beneš, P., and Sochor, J., "Computation of tunnels in protein molecules using Delaunay triangulation," in *Journal of WSCG*, pp. 107–114, 2007.

[78] Meyer, M., Desbrun, M., Schröder, P., and Barr, A. H., *Discrete Differential-Geometry Operators for Triangulated 2-Manifolds*, pp. 35–57. Heidelberg: Springer-Verlag, 2003. Hans-Christian Hege and Konrad Polthier, editors.

[79] Miura, T., Iwata, J., and Tsuda, J., "An application of hybrid curve generation: cartoon animation by electronic computers," in *AFIPS '67: Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, (New York, NY, USA), pp. 141–148, ACM, 1967.

[80] Mokhtarian, F., Abbasi, S., and Kittler, J., "Efficient and robust retrieval by shape content through curvature scale space," in *Proc. International Workshop IDB-MMS96*, pp. 35–42, 1996.

[81] Monge, G., *Applications de l'analyse à la géométrie*. Paris: Bachelier, fifth ed., 1894.

[82] Montanari, U., "Continuous skeletons from digitized images," *Journal of the Association for Computing Machinery*, vol. 16, no. 4, pp. 534–549, 1969.

[83] Mori, G., Belongie, S., and Malik, J., "Efficient shape matching using shape contexts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 11, pp. 1832–1837, 2005.

[84] Nackman, L., "Curvature relations in three-dimensional symmetric axes," *Computer Graphics & Image Processing*, vol. 20, pp. 43–57, 1982.

[85] Pasko, A., Adzhiev, V., Sourin, A., and Savchenko, V., "Function representation in geometric modeling: concepts, implementation and applications," *The Visual Computer*, vol. 11, no. 8, pp. 429–446, 1995.

[86] Pasko, G., Pasko, A., and Kunii, T. L., "Bounded blending for function-based shape modeling," *Computer Graphics and Applications, IEEE*, vol. 25, pp. 36–45, 2005.

[87] Pegna, J., *Variable Sweep Geometric Modeling*. PhD thesis, Mechanical Engineering Department, Stanford University, Palo Alto, CA, 1987.

[88] Pottmann, H., "Rational curves and surfaces with rational offsets," *Computed Aided Geometric Design*, vol. 12, pp. 175–192, 1995.

[89] Pratt, M. J., "Applications of cyclide surfaces in geometric modelling," in *Proceedings of the 3rd IMA Conference on the Mathematics of Surfaces*, (New York, NY, USA), pp. 405–428, Clarendon Press, 1989.

[90] Pudney, C., "Distance-ordered homotopic thinning: A skeletonization algoritm for 3D digital images," *IEEE. Trans. on Biomedical Engineering*, vol. 72, no. 3, pp. 404–413, 1998.

[91] Reeves, W. T., "Inbetweening for computer animation utilizing moving point constraints," *SIGGRAPH Comput. Graph.*, vol. 15, no. 3, pp. 263–269, 1981.

[92] Requicha, A. G. and Voelcker, H. B., "Constructive solid geometry," tech. rep., University of Rochester, 1977.

[93] Ricci, A., "A constructive geometry for computer graphics," *Comput. J.*, vol. 16, no. 2, pp. 157–160, 1973.

[94] Rockwood, A. P., "The displacement method for implicit blending surfaces in solid models," *ACM Trans. Graph.*, vol. 8, no. 4, pp. 279–297, 1989.

[95] Rockwood, A. and Owen, J., "Blending surfaces in solid modeling," in *Geometric Modelling: Algorithms and New Trends* (Farin, G., ed.), pp. 367–383, SIAM, 1986.

[96] Rossignac, J. and Requicha, A., "Constant-Radius Blending in Solid Modeling," *ASME Computers In Mechanical Engineering (CIME)*, vol. 3, pp. 65–73, 1984.

[97] Rossignac, J. and Requicha, A., "Piecewise-circular curves for geometric modeling," *IBM Journal of Research and Development*, vol. 13, pp. 296–313, 1987.

[98] Rossignac, J. and Kaul, A., "AGRELs and BIPs: Metamorphosis as a Bézier curve in the space of polyhedra," *Comput. Graph. Forum*, vol. 13, no. 3, pp. 179–184, 1994.

[99] Rossignac, J. and Requicha, A. A. G., "Piecewise-circular curves for geometric modeling," *IBM J. Res. Dev.*, vol. 31, no. 3, pp. 296–313, 1987.

[100] Sanglikar, M. A., Koparker, P., and Joshi, V. N., "Modeling rolling ball blends for computer aided geometric design," *Computer Aided Design*, vol. 7, pp. 399–414, 1990.

[101] Schmidt, R. and Wyvill, B., "Generalized sweep templates for implicit modeling," in *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, (New York, NY, USA), pp. 187–196, ACM, 2005.

[102] Schreiner, J., Asirvatham, A., Praun, E., and Hoppe, H., "Inter-surface mapping," in *SIGGRAPH 2004*, (Los Angeles, California), pp. 870–877, ACM, 2004.

[103] Sederberg, T. W., Gao, P., Wang, G., and Mu, H., "2-D shape blending: an intrinsic solution to the vertex path problem," in *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 15–18, ACM, 1993.

[104] Sederberg, T. W. and Greenwood, E., "A physically based approach to 2-D shape blending," in *SIGGRAPH '92: Proceedings of the 19th annual Conference on Computer Graphics and Interactive Techniques*, (New York, NY, USA), pp. 25–34, ACM, 1992.

[105] Serra, J., *Image Analysis and Mathematical Morphology*. New York: Academic Press, 1982.

[106] Shapira, M. and Rappoport, A., "Shape blending using the star-skeleton representation," *IEEE Comput. Graph. Appl.*, vol. 15, no. 2, pp. 44–50, 1995.

[107] Sherbrooke, E. C., Patrikalakis, N. M., and Brisson, E., "Computation of the medial axis transform of 3-D polyhedra," in *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications*, (New York, NY, USA), pp. 187–200, ACM, 1995.

[108] Siddiqi, K. and Pizer, S., *Medial Representations: Mathematics, Algorithms and Applications*. Springer, 2008.

[109] Soares, J., Leandro, J., Cesar, R., Jelinek, H., and Cree, M., "Retinal Vessel Segmentation Using the 2-D Gabor Wavelet and Supervised Classification," *IEEE Transactions on Medical Imaging*, vol. 25, no. 9, pp. 1214–1222, 2006.

[110] Sumner, R. W., Zwicker, M., Gotsman, C., and Popović, J., "Mesh-based inverse kinematics," in *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, (New York, NY, USA), pp. 488–495, ACM, 2005.

[111] Szymczak, A., Tannenbaum, A., and Mischaikow, K., "Coronary vessel cores from 3D imagery: a topological approach," in *Medical Imaging 2005: Image Processing. Proceedings of the SPIE*, vol. 5747, pp. 505–513, 2005.

[112] Taubin, G., "A signal processing approach to fair surface design," in *SIGGRAPH '95*, (New York, NY, USA), pp. 351–358, ACM, 1995.

[113] Telea, A. and van Wijk, J. J., "An augmented fast marching method for computing skeletons and centerlines," in *Symposium on Data Visualisation*, pp. 251–259, 2002.

[114] Tilove, R., "Set membership classification: A unified approach to geometric intersection problems," *IEEE Transactions on Computers*, vol. C-29, pp. 874–883, Oct. 1980.

[115] Tsai, V. J. D., "Fast topological construction of Delaunay triangulations and Voronoi diagrams," *Computers &.Geosciences*, vol. 19, no. 10, pp. 1463–1474, 1993.

[116] Tyrrell, J. A., di Tomaso, E., Fuja, D., Tong, R., Kozak, K., Jain, R., and Roysam, B., "Robust 3-D Modeling of Vasculature Imagery Using Superellipsoids," *IEEE Trans. on Medical Imaging*, vol. 26, no. 2, pp. 223–237, 2007.

[117] Van Dortmont, M., van de Wetering, H., and Telea, A., "Skeletonization and distance transforms of 3D volumes using graphics hardware," in *Discrete Geometry for Computer Imagery*, pp. 617–629, 2006.

[118] Várady, T., Vida, J., and Martin, R. R., "Parametric blending in a boundary representation solid modeller," in *Proceedings of the 3rd IMA Conference on the Mathematics of Surfaces*, (New York, NY, USA), pp. 171–197, Clarendon Press, 1989.

[119] Vida, K., Martin, R. R., and Varady, T., "A Survey of Blending Methods that Use Parametric Surfaces," *Compute Aided Design*, vol. 26, no. 5, pp. 341–365, 1994.

[120] Vinacua, A. and Rossignac, J., "SAM: Steady affine morph," *Submitted for review.*, 2009.

[121] Wang, S., Wang, Y., Jin, M., Gu, X., and Samaras, D., "3D surface matching and recognition using conformal geometry," in *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (Washington, DC, USA), pp. 2453–2460, IEEE Computer Society, 2006.

[122] Warren, J., "Blending algebraic surfaces," *ACM Transactions on Graphics*, vol. 8, no. 4, pp. 263–278, 1989.

[123] Weng, Y., Xu, W., Wu, Y., Zhou, K., and Guo, B., "2D shape deformation using nonlinear least squares optimization," *The Visual Computer*, vol. 22, no. 9, pp. 653–660, 2006.

[124] Whited, B., Rossignac, J., Slabaugh, G., Fang, T., and Unal, G., "Pearling: 3D interactive extraction of tubular structures from volumetric images," in *MICCAI Workshop: Interaction in Medical Image Analysis and Visualization*, 2007.

[125] Whited, B., Rossignac, J., Slabaugh, G., Fang, T., and Unal, G., "Pearling: Stroke segmentation with crusted pearl strings," *Pattern Recognition and Image Analysis*, vol. 19, pp. 277–283, 06 2009/06/01/.

[126] Whited, B. and Rossignac, J., "b-morphs between b-compatible curves in the plane," in *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, (New York, NY, USA), pp. 187–198, ACM, 2009.

[127] Whited, B. and Rossignac, J., "Relative blending," *Computer Aided Design*, vol. 41, no. 6, pp. 456–462, 2009.

[128] Wink, O., Niessen, W., and Viergever, M. A., "Fast delineation and visualization of vessels in 3-D angiographic images," *IEEE Trans. on Medical Imaging*, vol. 19, no. 4, pp. 337–346, 2000.

[129] Woodwark, J. R., "Blends in geometric modelling," in *Proceedings on Mathematics of surfaces II*, (New York, NY, USA), pp. 255–297, Clarendon Press, 1988.

[130] Wyvill, B., Galin, E., and Guy, A., "Extending The CSG Tree. Warping, Blending and Boolean Operations in an Implicit Surface Modeling System," *Computer Graphics Forum*, vol. 18, pp. 149–158, June 1999.

[131] Xu, D., Zhang, H., Wang, Q., and Bao, H., "Poisson shape interpolation," in *SPM '05: Proceedings of the 2005 ACM symposium on Solid and physical modeling*, (New York, NY, USA), pp. 267–274, ACM, 2005.

[132] Yang, Y., Brock, O., and Moll, R. N., "Efficient and robust computation of an approximated medial axis," in *SM '04: Proceedings of the ninth ACM symposium on Solid modeling and applications*, (Aire-la-Ville, Switzerland, Switzerland), pp. 15–24, Eurographics Association, 2004.

[133] Yezzi, A. J. and Prince, J. L., "An Eulerian PDE approach for computing tissue thickness," *IEEE Trans. Med. Imaging*, vol. 22, no. 10, pp. 1332–1339, 2003.

[134] Zhang, J. J. and You, L., "Surface representation using second, fourth and mixed order partial differential equations," in *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, (Washington, DC, USA), p. 250, IEEE Computer Society, 2001.

[135] Zhang, J. J. and You, L., "Rapid generation of C2 continuous blending surfaces," in *ICCS '02: Proceedings of the International Conference on Computational Science-Part II*, (London, UK), pp. 92–101, Springer-Verlag, 2002.